

*Finding dependencies in data with
information-theoretic methods*

A DISSERTATION PRESENTED

BY

Davor Sluga

TO

THE FACULTY OF COMPUTER AND INFORMATION SCIENCE

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN THE SUBJECT OF

COMPUTER AND INFORMATION SCIENCE



Ljubljana, 2017

*Finding dependencies in data with
information-theoretic methods*

A DISSERTATION PRESENTED

BY

Davor Sluga

TO

THE FACULTY OF COMPUTER AND INFORMATION SCIENCE

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN THE SUBJECT OF

COMPUTER AND INFORMATION SCIENCE



Ljubljana, 2017

APPROVAL

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

— Davor Sluga —

dr. Blaž Zupan

Full Professor of Computer and Information Science

EXAMINER

dr. Andrej Dobnikar

Full Professor of Computer and Information Science

EXAMINER

dr. Carsten Oliver Daub

Associate Professor of Bioinformatics

EXTERNAL EXAMINER

Karolinska Institutet, Department of Biosciences and Nutrition

dr. Marcin Iwanowski

Associate Professor of Computer Science

EXTERNAL EXAMINER

Warsaw University of Technology, Institute of Control and Industrial Electronics

dr. Uroš Lotrič

Associate Professor of Computer and Information Science

ADVISOR

PREVIOUS PUBLICATION

I hereby declare that the research reported herein was previously published/submitted for publication in peer reviewed journals or publicly presented at the following occasions:

- [1] D. Sluga, U. Lotrič Quadratic Mutual Information Feature Selection. *Entropy*, vol. 19(4), 157, 2017. doi: <http://dx.doi.org/10.3390/e19040157>
- [2] D. Sluga, T. Curk, B. Zupan and U. Lotrič Heterogeneous computing architecture for fast detection of SNP-SNP interactions. *BMC bioinformatics*, vol. 15, 1-16, 2014. doi: <http://dx.doi.org/10.1186/1471-2105-15-216>
- [3] D. Sluga, U. Lotrič Generalized information-theoretic measures for feature selection. In M. Tomassini, et al. *Adaptive and natural computing algorithms : proceedings*, 11th International Conference, ICANNGA 2013, Lausanne, Switzerland, 2013. doi: http://dx.doi.org/10.1007/978-3-642-37213-1_20
- [4] D. Sluga, T. Curk, B. Zupan, and U. Lotrič Acceleration of information-theoretic data analysis with graphics processing units. *Przegląd Elektrotechniczny (Electrical Review)*, vol. 88(2), 126–131, 2012. <http://pe.org.pl/articles/2012/2/38.pdf>

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Ljubljana.

Ko hodiš, pojdi zmeraj do konca.

Spomladi do rožne cvetice,
poleti do zrele pšenice,
jeseni do polne police,
pozimi do snežne kraljice,
v knjigi do zadnje vrstice,
v življenju do prave resnice,
a v sebi – do rdečice
čez eno in drugo lice.

A če ne prideš ne prvič, ne drugič
do krova in pravega kova
poskusi
vnovič
in zopet
in znova.

–Tone Pavček

POVZETEK

Izbira značilk je nujna na mnogih področjih, ki vključujejo visoko-dimenzionalne podatke. Izboljšuje možnost posploševanja regresijskih in klasifikacijskih metod in s tem tudi uspešnost modeliranja sistemov ter omogoča lažjo interpretacijo podatkov. Pomembnost značilk ocenimo s pomočjo kriterijske funkcije. Informacijske mere so samoumevne kandidatke, saj izhajajo iz cilja izbire značilk: želimo pridobiti nabor značilk, ki nudijo največ informacij o sistemu. Informacijsko-teoretične metode izbiranja značilk ponavadi potrebujejo oceno verjetnostne porazdelitve podatkov, na žalost pa se njeno ocenjevanje velikokrat izkaže za problematično. V disertaciji predlagamo novo metodo za izbiro značilk QMIFS (ang. quadratic mutual information feature selection), ki temelji na kvadratni medsebojni informaciji. Ta informacijsko-teoretična mera izhaja iz Cauchy-Schwarze divergence in Renyijeve entropije. Metoda uporablja neposredno oceno kvadratne medsebojne informacije iz podatkov z uporabo Gaussovih jedrnih funkcij. Zazna lahko nelinearne odvisnosti drugega reda. Izbira značilk iz velikih zbirk podatkov lahko postane zaradi dolgih izvajalnih časov praktično neuporabna. Računsko zahtevnost naše metode zmanjšamo s pomočjo nepopolne dekompozicije Choleskega nad vhodnimi podatki in ustreznim preoblikovanjem izračuna kriterijske funkcije. Učinkovitost predlagane metode se kaže skozi obsežno primerjavo z metodami MIFS (ang. mutual information feature selection), MRMR (ang. minimum redundancy maximum relevance) in JMI (ang. joint mutual information) na regresijski in klasifikacijski problemski domeni. Rezultati kažejo, da je predlagana metoda iz stališča uspešnosti modela primerljiva z ostalimi na klasifikacijskih problemih, le da je precej hitrejša. Na regresijskih problemih je nekoliko bolj uspešna od ostalih, vendar počasnejša. Zaporedni algoritmi za izbiro značilk lahko odpovejo, če podatkovna množica vsebuje več sto ali celo več tisoč značilk. Masovno vzporedni sistemi, kot na primer grafične procesne enote, nudijo veliko računske moči in naredijo analizo visoko-

dimenzionalnih podatkovnih množic časovno izvedljivo. Predlagano metodo za izbiro značilnk smo preoblikovali tako, da omogoča izrabo zmogljivih grafičnih procesnih enot in računskih koprosesorjev. Dosegli smo precejšnje zmanjšanje časov izvajanja, zaradi česar je metoda veliko bolj primerna za uporabo na velikih zbirkah podatkov.

Ključne besede: Iskanje značilnk, teorija informacij, Cauchy-Shwarzova divergenca, Renyijeva entropija, splošno namensko računanje na GPE

ABSTRACT

The selection of features that are relevant for a classification or a regression problem is very important in many domains which involve high-dimensional data. It improves the performance and generalization capabilities of regression and classification methods and facilitates the interpretation of the data about a system. To assess feature relevance, some kind of criterion function must be used. Information measures are an obvious candidate because they arise from the goal of feature selection: we wish to obtain a set of features that contain the most information about a system. Information-theoretic feature selection methods usually need to estimate the probability distributions of the data in order to assess feature relevance, this often proves to be problematic. In this thesis we propose a novel feature selection method (QMIFS) based on quadratic mutual information which has its roots in Cauchy–Schwarz divergence and Renyi entropy. The method uses the direct estimation of quadratic mutual information from data samples using Gaussian kernel functions, and can detect second order non-linear relations. Feature selection on large data sets can be infeasible due to long execution times. We reduce the computational complexity of the method through incomplete Cholesky decomposition of the input data and derive an appropriate estimator of the criterion function. The effectiveness of the proposed method is demonstrated through an extensive comparison with mutual information feature selection (MIFS), minimum redundancy maximum relevance (MRMR), and joint mutual information (JMI) on classification and regression problem domains. The experiments show that proposed method performs comparably to the other methods when applied to classification problems, except it is considerably faster. In the case of regression, it compares favourably to the others, but is slower. If the data includes hundreds or even thousands of features, sequential algorithms for feature selection may no longer suffice. Nowadays massively parallel systems such as graphics processing units offer the computational power to make anal-

ysis of high-dimensional data feasible. We modified the proposed method to make use of the powerful hardware of graphics processing units. We achieve considerable improvements in the execution times, making it viable for usage on large data sets.

Key words: feature selection, information theory, Cauchy-Shwarz divergence, Renyi entropy, general-purpose GPU computing

CONTENTS

	<i>Povzetek</i>	<i>i</i>
	<i>Abstract</i>	<i>iii</i>
1	<i>Introduction</i>	<i>I</i>
1.1	Contributions to science	3
1.2	Thesis outline	4
2	<i>Information theory</i>	<i>7</i>
2.1	Shannon measures of information	8
2.1.1	Kullback-Leibler divergence	9
2.1.2	Mutual information	10
2.1.3	Mutual information between multiple random variables	11
2.1.4	Differential information-theoretic measures	13
2.2	Generalizations of Shannon information-theoretic measures	14
2.2.1	Renyi divergence and mutual information	14
2.2.2	Differential Renyi entropy	15
2.3	Estimation of information-theoretic measures	16
2.3.1	Non-parametric Parzen-window estimation	16
2.3.2	Quadratic Renyi entropy	18
2.3.3	Cauchy-Schwarz divergence	18
2.3.4	Quadratic mutual information	19
3	<i>Feature selection</i>	<i>21</i>
3.1	Feature relevancy	23

3.2	The process of feature selection	23
3.3	Feature selection methods	25
3.3.1	Wrapper methods	25
3.3.2	Embedded methods	25
3.3.3	Filter methods	25
3.4	Information-theoretic feature selection	26
4	<i>Proposed method</i>	29
4.1	Quadratic mutual information feature selection - QMIFS	31
4.2	Performance on artificial data sets	37
4.3	Results and discussion	38
4.3.1	Experimental methodology	41
4.3.2	Classification performance	43
4.3.3	Regression performance	52
5	<i>Optimizations of QMIFS</i>	61
5.1	Incomplete Cholesky decomposition	62
5.2	Exploiting custom hardware	68
5.2.1	Parallel computing on GPUs	69
5.2.2	Parallel computing on the MIC coprocessors	79
5.2.3	QMIFS on a heterogenous architecture	82
5.3	Results and discussion	83
6	<i>Conclusion</i>	95
6.1	Future work	97
A	<i>Supplement to chapter "Proposed method"</i>	99
B	<i>Razširjeni povzetek</i>	117
B.1	Uvod	118
B.1.1	Prispevki k znanosti	119
B.2	Teorija informacij	120
B.3	Izbira značilk	121
B.4	Predlagana metoda	123
B.5	Izboljšave QMIFS	124

B.6	Zaključek	126
	<i>Bibliography</i>	<i>129</i>

Introduction

High-dimensional data are present in many areas of engineering and research like image and signal processing, biological and medical data analysis, and advertising. The availability of low cost sensors, increased capacity and low costs of storage equipment along with ability to transfer large amounts of data seamlessly across the world facilitates the concurrent measurement of many features. The idea is that adding features can only increase the amount of information about a system. The catch is that high-dimensional data are in general more difficult to analyze. Many data analysis tools fail when confronted with high-dimensional data, or provide meaningless results. Problems that arise due to handling high-dimensional data are frequently referred to as the curse of dimensionality [1].

To counteract these difficulties two approaches are possible. One is to develop tools that are able to model high-dimensional data in a lower-dimensional space, like for example support-vector machines. The other is to reduce the dimensionality of the data without throwing away too much information. There are two possible ways to reduce the dimensionality of the data: feature extraction, which produces data of lower dimension by combining features together, or feature selection, which selects a subset of the original features and discards the others by means of some criterion function. The latter approach has the advantage of preserving the original features, which may help a data analyst interpret the model or analysis results.

There are many types of criterion functions that can be used to assess the relevancy of the features. They are based on different measures and their extensions [2]. These include measures of distance or divergence, dependency measures, consistency measures, and information measures [3].

Information measures are definitely among the most interesting because they arise from the definition of feature selection: we wish to obtain a set of features providing the largest amount of information about the target system. Commonly used measures include average mutual information [4] together with its extensions and relative entropy. Measures based on generalizations of Shannon entropy, for example Renyi [5] and Tsallis [6] entropy, are also possible candidates. All information measures share the necessity to calculate the probability distribution of the data. This is usually done by building histograms or using kernel density estimation. These methods can be problematic from two aspects: the computational complexity and the accuracy of approximation of the probability distributions; especially when dealing with high-dimensional probability distributions. Most information-theoretic feature selection methods deal

with discrete data only. This means that if data includes continuous features, they need to be discretized beforehand, which adds an additional step to the whole process, possibly shrouding some intricate details in the data. A partial solution to this problem can be found in the work of Principe et al. [7], where Renyi entropy in combination with Parzen windows is used to efficiently calculate the quadratic mutual information directly from the data (continuous or discrete), without the intermediate step of probability density estimation.

Computational complexity of feature selection methods emerges as an issue when dealing with large data sets. Feasible execution times can be achieved by considering modern massively parallel architectures like graphics processing units capable of general purpose computing and providing much more computing power than central processing units [8]. They can be used to speed up the computations, but only if the problem solving algorithm can be efficiently parallelized [9]. Feature selection usually involves computing some criterion function on a multitude of features or feature subsets. The computations are independent of each other, which makes parallelization of feature selection methods viable. This is especially important if the criterion function is computationally expensive and we need to analyze large datasets, like genetic data, where the number of features can reach millions [10].

1.1 Contributions to science

The main topic of this thesis is introduction of a new information-theoretic measure for finding relevant features and searching for dependencies in data, both from the perspective of relevancy of the selected features and faster processing. We primarily focus on detecting interactions between pairs of features, which can potentially better explain the observed data. Furthermore, we want to avoid preprocessing steps like discretization of the data or estimation of probability distributions and cope with data directly. We summarize our contributions to the scientific community in the following.

- *Development of a novel feature selection method based on extended information-theoretic measures.* We propose the quadratic mutual information feature selection algorithm (QMIFS) that uses an evaluation criterion based on quadratic mutual information to guide the feature selection process. The proposed method uses a greedy search strategy to sweep through the feature space and is able to detect second-order non-linear relations between features and the output. It can

deal with discrete or continuous features and output, which makes it suitable to use without any preprocessing dependent on expert knowledge about the data. Moreover, it avoids using parameters, which are inconvenient for non-experts in the field. It is possible to use it as a precursor to classification as well as regression problems in order to avoid over-fitting and to improve the learning machine performance.

- *Optimization of computing the criterion function using incomplete Cholesky decomposition.* We integrate incomplete Cholesky decomposition (ICD) into the criterion function of QMIFS to reduce its computational complexity from quadratic to linear with regards to the number of instances in the data. To achieve this we derive an estimator for the three-variate quadratic mutual information that uses incomplete Cholesky decomposition of the input variables.
- *Parallelization of the developed method for massively parallel systems.* We introduce appropriate modifications of QMIFS to make it feasible for execution on massively parallel architectures. We identify the most computationally intensive part of the QMIFS algorithm and adopt it for execution on a massively parallel architectures – graphics processing units (GPUs) and Many core architecture (MIC) coprocessors. In the process we take special care in minimizing the memory transfer overhead and unnecessary coordination between the CPU and the GPU/MIC coprocessor. Furthermore, we investigate the possibility of using all of the processing resources simultaneously to further reduce the running time of the method.

1.2 Thesis outline

We begin the thesis with a short introduction to information theory. In Chapter 2 we familiarize with the information-theoretic terminology and put forward common information-theoretic quantities. We also present some extensions to the original concepts of information theory and summarize the approaches for estimation of information-theoretic quantities. In Chapter 3 we review the field of feature selection, with emphasis on methods based on information-theoretic measures. We proceed with the proposal of a novel feature selection method (QMIFS) in Chapter 4. The method is based on quadratic mutual information and uses a greedy search algorithm to find relevant features. We compare QMIFS to three other information-theoretic features on

multiple datasets. In Chapter 5 we improve the method using incomplete Cholesky decomposition. We familiarize with the concept of using GPUs for general-purpose computing and propose a modification of QMIFS suitable for execution on a GPU. We further extend the concept to the MIC architecture and to combination of CPU, GPU, and MIC. We test the different QMIFS implementations on multiple data sets and compare them with the original proposal and other comparable methods with regards to execution time. We summarize the thesis in Chapter 6 with further research directions in mind.



Information theory

Information theory is a science field dealing with quantification, storage, and communication of information [11]. Its origins can be traced back to the pioneering works of Nyquist who in 1927 worked on the problem of sending information over a telegraph channel with maximum possible speed and without distortion. In 1928 Hartley further developed the field when he tried to define a measure of information. The foundations of information theory as we know it today were laid down in 1948 by Shannon [12] when he extended the works of Nyquist and Hartley with the concepts of entropy and probability.

2.1 Shannon measures of information

Let X be a discrete scalar random variable which can acquire values $\{x_1, \dots, x_n\}$ with probabilities $\{p_X(x_1), \dots, p_X(x_n)\}$. For the discrete probability mass function p_X it holds that $p_X(x_i) \geq 0$, $i = 1, \dots, n$, and $\sum_{i=1}^n p_X(x_i) = 1$. The entropy $H(X)$ is then defined as [4]

$$H(X) = - \sum_{i=1}^n p_X(x_i) \log p_X(x_i). \quad (2.1)$$

The base of the algorithm is arbitrary. However, the usual approach is to use base 2, which yields entropy measured in *bits*. Entropy can be interpreted as the average amount of information one has to obtain about a random variable in order to remove all uncertainty about its value [13].

Entropy is minimal, $H(X) = 0$, when there is no uncertainty about the value of X – its probability distribution is deterministic. It obtains the maximum value $H(X) = \log n$ when the probabilities are equal for every possible value of X , in such case the probability mass function is equal to $p_X = \left\{ \dots, \frac{1}{n}, \dots \right\}$.

We can extend the notion of entropy to a vector random variable $\mathbf{X} = (X_1, \dots, X_D)$, where X_1, \dots, X_D are scalar random variables. Vector random variable \mathbf{X} can then acquire values $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with probabilities $\{p_{\mathbf{X}}(\mathbf{x}_1), \dots, p_{\mathbf{X}}(\mathbf{x}_n)\}$. Entropy of \mathbf{X} is

$$H(\mathbf{X}) = - \sum_{i=1}^n p_{\mathbf{X}}(\mathbf{x}_i) \log p_{\mathbf{X}}(\mathbf{x}_i). \quad (2.2)$$

If we introduce another scalar random variable Y , which can acquire values $\{y_1, \dots, y_m\}$ with probabilities $\{p_Y(y_1), \dots, p_Y(y_m)\}$, we can express its entropy as

$$H(Y) = - \sum_{i=1}^m p_Y(y_i) \log p_Y(y_i) . \quad (2.3)$$

One can also observe the ambiguity of both \mathbf{X} and Y when observed together. We can compute the joint entropy $H(\mathbf{X}, Y)$ by considering the joint probabilities $p_{\mathbf{X}Y}(\mathbf{x}_i, y_j)$ of events in which \mathbf{X} will take on the value \mathbf{x}_i , and Y the value y_j , $i = 1, \dots, n$, $j = 1, \dots, m$. From this we can deduce the equation for computing the joint entropy

$$H(\mathbf{X}, Y) = - \sum_{i,j} p_{\mathbf{X}Y}(\mathbf{x}_i, y_j) \log p_{\mathbf{X}Y}(\mathbf{x}_i, y_j) . \quad (2.4)$$

If the two variables are independent then it holds that $p_{\mathbf{X}Y}(\mathbf{x}_i, y_j) = p_{\mathbf{X}}(\mathbf{x}_i)p_Y(y_j)$, and the joint entropy is equal to the sum of entropies, $H(\mathbf{X}, Y) = H(\mathbf{X}) + H(Y)$.

Furthermore, we can also define the conditional entropy of \mathbf{X} when Y takes on the value y_j ,

$$H(\mathbf{X}|Y = y_j) = - \sum_i p_{\mathbf{X}|Y}(\mathbf{x}_i, y_j) \log p_{\mathbf{X}|Y}(\mathbf{x}_i, y_j) , \quad (2.5)$$

where $p_{\mathbf{X}Y}(\mathbf{x}_i, y_j) = p_{\mathbf{X}|Y}(\mathbf{x}_i, y_j)p_Y(y_j)$, and $p_{\mathbf{X}|Y}(\mathbf{x}_i, y_j)$ is the conditional probability of \mathbf{X} taking on the value of \mathbf{x}_i , given the value of $Y = y_j$. The average of conditional entropies across all y_j , is the conditional entropy

$$H(\mathbf{X}|Y) = - \sum_j p_Y(y_j) H(\mathbf{X}|Y = y_j) , \quad (2.6)$$

which gives us the entropy of \mathbf{X} when we already know the value of Y .

2.1.1 Kullback-Leibler divergence

Another important information-theoretic measure is the Kullback-Leibler divergence (KL) that measures discrepancy between two probability mass functions $p_{\mathbf{X}}$ and $p'_{\mathbf{X}}$ defined over the same random variable.

$$D_{\text{KL}}(p_{\mathbf{X}}; p'_{\mathbf{X}}) = \sum_{i=1}^n p_{\mathbf{X}}(\mathbf{x}_i) \log \frac{p_{\mathbf{X}}(\mathbf{x}_i)}{p'_{\mathbf{X}}(\mathbf{x}_i)} . \quad (2.7)$$

It can be sought as a generalization of algebraic distance measures, such as Euclidean norm, to probability distributions. It is always non-negative and is zero if and only if $p_{\mathbf{X}}$ and $p'_{\mathbf{X}}$ are equal. However, it must be noted that KL divergence is not a true metric, since it does not obey triangle inequality and in general is not symmetric – $D_{\text{KL}}(p_{\mathbf{X}}; p'_{\mathbf{X}})$ does not equal $D_{\text{KL}}(p'_{\mathbf{X}}; p_{\mathbf{X}})$.

The extension of the KL divergence to a pair of random variables (\mathbf{X}, Y) yields

$$D_{\text{KL}}(p_{\mathbf{X}Y}; p'_{\mathbf{X}Y}) = \sum_{i,j} p_{\mathbf{X}Y}(\mathbf{x}_i, y_j) \log \frac{p_{\mathbf{X}Y}(\mathbf{x}_i, y_j)}{p'_{\mathbf{X}Y}(\mathbf{x}_i, y_j)}, \quad (2.8)$$

where $p_{\mathbf{X}Y}$ and $p'_{\mathbf{X}Y}$ are two joint probability distributions defined over \mathbf{X} and Y .

2.1.2 Mutual information

Mutual information (MI) measures the amount of information one random variable contains about another random variable. Lets consider a pair of random variables (\mathbf{X}, Y) . Mutual information $I(\mathbf{X}; Y)$ is then the KL divergence between the joint distribution $p_{\mathbf{X}Y}$ and the product distribution $p'_{\mathbf{X}Y} = p_{\mathbf{X}}p_Y$, which assumes the two variables are independent

$$I(\mathbf{X}; Y) = D_{\text{KL}}(p_{\mathbf{X}Y}; p_{\mathbf{X}}p_Y) \quad (2.9)$$

$$= \sum_{i,j} p_{\mathbf{X}Y}(\mathbf{x}_i, y_j) \log \frac{p_{\mathbf{X}Y}(\mathbf{x}_i, y_j)}{p_{\mathbf{X}}(\mathbf{x}_i)p_Y(y_j)} \quad (2.10)$$

$$= H(\mathbf{X}) + H(Y) - H(\mathbf{X}, Y) \quad (2.11)$$

$$= H(\mathbf{X}) - H(\mathbf{X}|Y). \quad (2.12)$$

Mutual information is non-negative and equal zero if the two random variables are independent. If \mathbf{X} is completely dependent on Y then it holds that $H(\mathbf{X}|Y) = 0$, since Y unambiguously defines the value of \mathbf{X} . In this case mutual information is maximal and equal to $H(\mathbf{X})$. Due to symmetry, it also follows that

$$I(\mathbf{X}; Y) = H(Y) - H(Y|\mathbf{X}). \quad (2.13)$$

Thus \mathbf{X} says as much about Y as Y says about \mathbf{X} . The relations between the measures of entropy and mutual information are most clearly presented using a Venn diagram in Fig. 2.1.

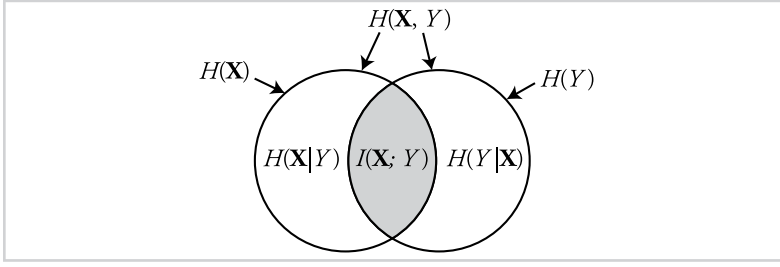


Figure 2.1

Venn diagram representing relations between $H(\mathbf{X})$, $H(Y)$, $H(\mathbf{X}, Y)$, $H(\mathbf{X}|Y)$, $H(Y|\mathbf{X})$, and $I(\mathbf{X}; Y)$.

2.1.3 Mutual information between multiple random variables

Various generalizations of mutual information to more than two random variables have been proposed [14]. One approach is to derive it directly from the KL divergence similarly to the two-variable case. Let's again consider the vector random variable $\mathbf{X} = (X_1, \dots, X_D)$. To measure the amount of information that is shared between variables in the set, we can compute the KL divergence between $p_{\mathbf{X}}$ and the product distribution $\prod_{k=1}^D p_{X_k}$,

$$C(X_1, \dots, X_D) = D_{\text{KL}}(p_{\mathbf{X}}; \prod_{k=1}^D p_{X_k}) \quad (2.14)$$

$$= \sum_{k=1}^D H(X_k) - H(\mathbf{X}). \quad (2.15)$$

The quantity $C(X_1, \dots, X_D)$ is called total correlation [15]. In the case where the vector random variable consists of only two variables $\mathbf{X} = (X_1, X_2)$, the total correlation $C(X_1, X_2)$ is equal to the mutual information $I(X_1; X_2)$.

Another way towards the definition of mutual information for multiple variables is to first consider the three-variate case. Besides X and Y let us now introduce a new scalar random variable Z with probability mass function p_Z . We could now try to find out how introducing this new variable into our environment affects the mutual information between X and Y . We can approach this problem by using the three-variate Venn diagram. The red area in Fig. 2.2 shows $I(X; Y|Z)$, the conditional mutual information between X and Y , given the knowledge of Z . From the Venn diagram we can easily deduce an equation which gives $I(X; Y|Z)$ using already familiar quantities.

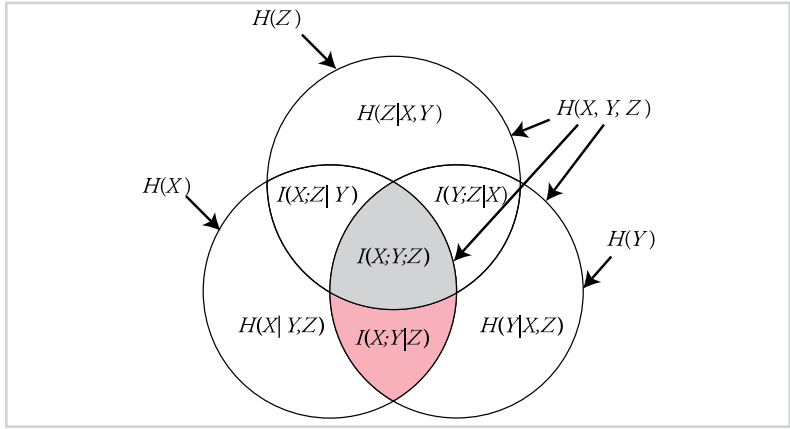


Figure 2.2

Venn diagram representing relations between information-theoretic quantities in the three-variate case.

One possibility is

$$I(X; Y|Z) = H(X|Z) - H(X|Y, Z). \quad (2.16)$$

Conditional mutual information is non-negative and is zero if X and Y are unrelated given the knowledge of Z , or if Z completely describes the connection between X and Y . Conditional mutual information $I(X; Y|Z)$ thus explains the relation between X and Y in the presence of Z , but it does not measure the amount of information brought into the environment by introducing Z . This can be measured by observing the intersection of all three variables in Fig. 2.2 – the area marked in grey. This is the multivariate mutual information $I(X; Y; Z)$ between all three variables

$$\begin{aligned} I(X; Y; Z) &= I(X; Y) - I(X; Y|Z) \\ &= I(X; Z) + I(Y; Z) - I(X, Y; Z). \end{aligned} \quad (2.17)$$

Multivariate mutual information expresses the amount of information common to a set of variables, but not present in any subset. It is symmetric, but unlike mutual information, it can be either positive or negative. In some works authors put a negative sign in front of the quantity and call it interaction information [14]. Negative value of multivariate mutual information indicates that variable Z explains some of the relation between X and Y , whereas a positive value suggests that Z provides redundant information about the relation between X and Y .

2.1.4 Differential information-theoretic measures

In the previous section we were dealing with discrete random variables, which can only take on a finite amount of values. If the set of possible values is infinite, we call this kind of variable a continuous random variable. In this case the distribution of values is described using a continuous probability density function $p_X(x)$ for which it holds that $\int p_X(x)dx = 1$.

The differential entropy $H^d(X)$ of a continuous random variable with density $p_X(x)$ is then defined as

$$H^d(X) = - \int p_X(x) \log p_X(x) dx . \quad (2.18)$$

Similarly as in Eq. 2.2 we can extend the notion of entropy to a vector of continuous random variables, where $p_{\mathbf{X}}(\mathbf{x})$ is the joint probability density function,

$$H^d(\mathbf{X}) = - \int p_{\mathbf{X}}(\mathbf{x}) \log p_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} . \quad (2.19)$$

Unlike discrete entropy, differential entropy can be negative as shown in the following example. Consider an uniformly distributed random variable on the interval $[0, a]$. The probability density in the interval is equal to $1/a$, and 0 elsewhere. Following Eq. 2.19 we obtain

$$H^d(\mathbf{X}) = - \int_0^a \log dx = \log a , \quad (2.20)$$

which yields a negative differential entropy for $0 < a < 1$.

Kullback-Leibler divergence and mutual information can also be extended to probability density functions. Differential KL divergence between two probability densities $p_{\mathbf{X}}$ and $p'_{\mathbf{X}}$ is defined by

$$D_{\text{KL}}^d(p_{\mathbf{X}}, p'_{\mathbf{X}}) = \int p_{\mathbf{X}}(\mathbf{x}) \log \frac{p_{\mathbf{X}}(\mathbf{x})}{p'_{\mathbf{X}}(\mathbf{x})} d\mathbf{x} . \quad (2.21)$$

From there, as in Eq. 2.9, we also define differential mutual information between two continuous random variables \mathbf{X} and Y as

$$I^d(\mathbf{X}; Y) = D_{\text{KL}}^d(p_{\mathbf{X}Y}, p_{\mathbf{X}}p_Y) . \quad (2.22)$$

The properties of differential KL divergence and mutual information are the same as in the discrete case.

2.2 Generalizations of Shannon information-theoretic measures

Besides the classical Shannon entropy, there exist a range of information entropy generalizations [16]. One of the more widely known is the Renyi Entropy [17]. Renyi tried to establish the most general definition of information measures that would preserve the additivity of independent events and compatibility with Kolmogorov axioms of probability. He extended the ordinary mean in Eq. 2.1 with an exponential class of functions and obtained a parametric family of information measures – Renyi entropies. In case of a discrete vector random variable \mathbf{X} , Renyi entropy of order q is given by

$$H_{R_q}(\mathbf{X}) = \frac{1}{1-q} \log \sum_{i=1}^n p_{\mathbf{X}}(\mathbf{x}_i)^q, \quad (2.23)$$

with $q > 0$ and $q \neq 1$. It should be noted that Renyi entropy converges to Shannon entropy as q approaches 1 in the limit [4].

From Eq. 2.23 we can establish a few key properties of Renyi's entropies:

- $H_{R_q}(\mathbf{X})$ is non-negative, and is equal to zero if and only if the distribution is degenerated,
- $H_{R_q}(\mathbf{X})$ obtains the maximum value of $H_{R_q}(\mathbf{X}) = \log n$, when $p_{\mathbf{X}}(\mathbf{x}_i) = \frac{1}{n}$, $\forall i \in \{1, \dots, n\}$,
- $H_{R_q}(\mathbf{X})$ is monotonous and non increasing function of q ,
- $H_{R_q}(\mathbf{X}, Y) = H_{R_q}(\mathbf{X}) + H_{R_q}(Y)$, if and only if \mathbf{X} and Y are independent.

2.2.1 Renyi divergence and mutual information

Since Renyi entropy is a generalization of Shannon entropy, one can also establish a generalization of the Kullback-Leibler divergence between two discrete probability distributions $p_{\mathbf{X}}$ and $p'_{\mathbf{X}}$. Renyi [17] proposed the following definition

$$D_q(p_{\mathbf{X}}; p'_{\mathbf{X}}) = \frac{1}{1-q} \log \sum_{i=1}^n \frac{p_{\mathbf{X}}(\mathbf{x}_i)^q}{p'_{\mathbf{X}}(\mathbf{x}_i)^{q-1}}. \quad (2.24)$$

The key properties of the Renyi divergence are the following

- $D_q(p_{\mathbf{X}}; p'_{\mathbf{X}}) \geq 0$, $\forall q > 0$,

- $D_q(p_{\mathbf{X}}; p'_{\mathbf{X}}) = 0$, if and only if $p_{\mathbf{X}} = p'_{\mathbf{X}}$,
- $\lim_{q \rightarrow 1} D_q(p_{\mathbf{X}}; p'_{\mathbf{X}}) = D_{\text{KL}}(p_{\mathbf{X}}, p'_{\mathbf{X}})$.

In the previous section we derived the notion of mutual information from the definition of Kullback-Leibler divergence between the joint distribution $p_{\mathbf{X}Y}$ and the product distribution $p_{\mathbf{X}}p_Y$. Similarly, Renyi q mutual information can be derived in the same way as

$$I(\mathbf{X}; Y)_q = D_q(p_{\mathbf{X}Y}, p_{\mathbf{X}}p_Y) = \frac{1}{1-q} \log \sum_{i,j} \frac{p_{\mathbf{X}Y}(\mathbf{x}_i, y_j)^q}{(p_{\mathbf{X}}(\mathbf{x}_i)p_Y(y_j))^{q-1}}. \quad (2.25)$$

Again, in the limit as q approaches 1, $I(\mathbf{X}; Y)_q$ becomes Shannon mutual information. However, $I(\mathbf{X}; Y)_q$ does not follow all of the properties of Shannon mutual information. It is non-negative and symmetric but it can yield strange results. For example, information on Y given by \mathbf{X} can be larger than the information of \mathbf{X} . This means that the Venn diagram representation of relations between information-theoretic quantities does not hold in the case of Renyi's generalizations. Renyi tried different approaches to define a generalized measure of mutual information, but unfortunately all of them have shortcomings [17].

2.2.2 Differential Renyi entropy

To deal with continuous random variables, Renyi also proposed the differential Renyi Entropy. Let $p_{\mathbf{X}}(\mathbf{x})$ be an integrable probability density function, then the differential Renyi entropy of order q is

$$H_{R_q}^d(\mathbf{X}) = \frac{1}{1-q} \log \int p_{\mathbf{X}}(\mathbf{x})^q d\mathbf{x}. \quad (2.26)$$

Similarly to the differential Shannon entropy, the differential Renyi entropy can be negative for $q < 1$. It becomes Shannon differential entropy when q approaches 1. Analogously to the differential entropy, both differential Renyi divergence and mutual information can be established by replacing the summation with an integral over probability density functions

$$D_q^d(p_{\mathbf{X}}, p'_{\mathbf{X}}) = \frac{1}{1-q} \log \int \frac{p_{\mathbf{X}}(\mathbf{x})^q}{p'_{\mathbf{X}}(\mathbf{x})^{q-1}} d\mathbf{x}, \quad (2.27)$$

and

$$I_q^d(\mathbf{X}; Y) = D_q^d(p_{\mathbf{X}Y}, p_{\mathbf{X}}p_Y) . \quad (2.28)$$

2.3 *Estimation of information-theoretic measures*

Obviously, there is a difference on how to estimate information-theoretic measures if we are dealing with discrete or continuous variables. If the variables are discrete, estimation involves finding the probability distribution using histograms and then performing summations over the probabilities.

When the variables are continuous, one option is to apply a discretization step beforehand (equal width binning, equal frequency binning, etc.) [18] and again use summations. The manual selection of the number of bins can affect estimation of MI and can lead to spurious results by shrouding some properties of the probability distribution. A better approach is to perform the discretization using an adaptive technique like minimum description length (MDL) [19].

The more direct approach when dealing with continuous variables is to use differential information-theoretic measures. Usually, the estimation includes a probability density function (PDF) estimation from the data followed by the integral estimation from the PDF. Other approaches to differential information-theoretic measures estimation have also been proposed in the literature, due to the problematic nature of PDF estimation in high dimensional space and computationally expensive numerical integration. One of them is the kNN estimator [20] of MI, which in certain situations provides better results than the Parzen-window, but is still computationally expensive and not suitable to use directly on data sets comprised of discrete and continuous data [21]. More recent approach is to estimate the density ratio (2.21) directly. However, due to the logarithm in Eq. 2.21, this approach becomes computationally expensive and susceptible to outliers [22]. To alleviate this problem, the authors [22] propose squared-loss mutual information measure which makes the computation more robust.

2.3.1 *Non-parametric Parzen-window estimation*

When dealing with continuous variables, in order to compute the entropy or mutual information, the most straightforward approach is to use the non-parametric Parzen

window estimator for the PDF. Given n independent and identically distributed samples $\{x_1, \dots, x_n\}$ of a random variable X , the Parzen estimate of the PDF is

$$\hat{p}_X(x) = \frac{1}{n} \sum_{i=1}^n \kappa_\sigma(x - x_i), \quad (2.29)$$

where κ_σ is the kernel function. The subscript σ defines the width of the kernel. The kernel function must have the following proprieties [23]:

- $\kappa_\sigma(x) \geq 0$,
- $\int_R \kappa_\sigma(x) dx = 1$,
- $\lim_{x \rightarrow \infty} |x \kappa_\sigma(x)| = 0$.

The kernel functions normally used are symmetric with the peak on the sample, continuous, and differentiable. The Gaussian kernel is a common choice

$$G_\sigma(x - x_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-x_i)^2/2\sigma^2}. \quad (2.30)$$

The kernel width σ is a free parameter in the Parzen-window PDF estimation. Multiple recipes exist on how to set it appropriately [24]. One of the most commonly used is the Silverman rule of thumb [23]

$$\sigma = 1.06 \sigma_X n^{-0.2}, \quad (2.31)$$

where σ_X is the standard deviation of random variable X .

When dealing with vector random variables, the Gaussian kernel must be extended to multi-dimensional space. This can be achieved through the product of one-dimensional Gaussian kernels

$$\mathbf{G}_\sigma(\mathbf{x} - \mathbf{x}_i) = \prod_{k=1}^D G_{\sigma_k}(x_k - x_{ki}), \quad (2.32)$$

where x_{ki} is the i -th sample of variable X_k and $\sigma = \{\sigma_1, \dots, \sigma_D\}$.

2.3.2 Quadratic Renyi entropy

Erdogmus et al. [25] bypassed the intermediate step of PDF estimation and showed that quadratic Renyi entropy ($q = 2$) can be directly estimated from the data, without the explicit need to estimate the PDF. By assuming Gaussian kernels with kernel σ and substituting the PDF with the Parzen-window estimate in Eq. 2.26 we get

$$\begin{aligned}
 \hat{H}_{R_2}^d(X) &= -\log \int_{-\infty}^{\infty} \left(\frac{1}{n} \sum_{i=1}^n G_{\sigma}(x - x_i) \right)^2 dx \\
 &= -\log \frac{1}{n^2} \int_{-\infty}^{\infty} \left(\sum_{i=1}^n \sum_{j=1}^n G_{\sigma}(x - x_i) G_{\sigma}(x - x_j) \right) dx \\
 &= -\log \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \int_{-\infty}^{\infty} G_{\sigma}(x - x_i) G_{\sigma}(x - x_j) dx \\
 &= -\log \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2}\sigma}(x_i - x_j) .
 \end{aligned} \tag{2.33}$$

$$(2.34)$$

The result is obtained by observing that the integral of the product of two Gaussian functions is a Gaussian function whose variance is the sum of the variances of the two original Gaussian functions. Note, however, that other kernel functions do not result in such handy evaluation of the integral.

Derivation for a vector random variable \mathbf{X} follows along similar lines, but instead of the Gaussian kernel we use the product of Gaussian functions given in Eq. 2.32. This yields the estimate of quadratic Renyi entropy for a vector random variable

$$\hat{H}_{R_2}^d(\mathbf{X}) = -\log \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2}\sigma}(\mathbf{x}_i - \mathbf{x}_j) . \tag{2.35}$$

2.3.3 Cauchy-Schwarz divergence

On the basis of Renyi entropy and it's convenient estimation the same authors [7] also proposed a new divergence measure between two PDFs called Cauchy-Schwartz

divergence derived from the Cauchy-Schwarz inequality [26]:

$$\int f^2(x) \int g^2(x) dx \geq \left(\int f(x)g(x) dx \right)^2, \quad (2.36)$$

where $f(x)$ and $g(x)$ are square-integrable functions. Equality holds if and only if $f(x) = cg(x)$ for a constant scalar c . From there they define the Cauchy-Schwarz divergence as

$$D_{\text{CS}}(p_{\mathbf{X}}, p'_{\mathbf{X}}) = -\log \frac{\left(\int p_{\mathbf{X}}(\mathbf{x}) p'_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} \right)^2}{\int p_{\mathbf{X}}^2(\mathbf{x}) d\mathbf{x} \int p'^2_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}}. \quad (2.37)$$

The divergence is always non-negative and is equal to zero if and only if $p_{\mathbf{X}}$ equals $p'_{\mathbf{X}}$, which is the same as in the case of KL divergence and Renyi divergence. Furthermore, $D_{\text{CS}}(p_{\mathbf{X}}, p'_{\mathbf{X}})$ is symmetric, but it doesn't obey the triangle inequality. Lets assume $p_{\mathbf{X}}$ represents the PDF of a continuous vector random variable \mathbf{X} and p_Y the PDF of a continuous random variable Y . We can rearrange the above equation and write it in terms of quadratic Renyi entropy as

$$D_{\text{CS}}(p_{\mathbf{X}}, p_Y) = 2H_{\text{R}_2}^d(\mathbf{X}; Y) - H_{\text{R}_2}^d(\mathbf{X}) - H_{\text{R}_2}^d(Y), \quad (2.38)$$

where the first term

$$H_{\text{R}_2}^d(\mathbf{X}; Y) = -2 \log \int \int p_{\mathbf{X}}(\mathbf{x}) p_Y(y) d\mathbf{x} dy, \quad (2.39)$$

is the quadratic Renyi cross-entropy [27]. It can be directly estimated from the data using a similar approach as in the case of $\hat{H}_{\text{R}_2}^d(\mathbf{X})$

$$\hat{H}_{\text{R}_2}^d(\mathbf{X}; Y) = -\log \frac{1}{n^{D+2}} \sum_{i=1}^n \left[\left(\sum_{l=1}^n G_{\sqrt{2}\sigma} (y_i - y_l) \right) \right]. \quad (2.40)$$

$$\prod_{k=1}^D \left(\sum_{j=1}^n G_{\sqrt{2}\sigma_k} (x_{ki} - x_{kj}) \right) \right]. \quad (2.41)$$

2.3.4 Quadratic mutual information

On the basis of Cauchy-Schwarz divergence (2.38), Principe et al. [7] proposed quadratic mutual information (QMI)

$$I_{\text{CS}}(\mathbf{X}; Y) = D_{\text{CS}}(p_{\mathbf{X}Y}, p_{\mathbf{X}} p_Y) \quad (2.42)$$

as a candidate for measuring dependence between variables. They prove that $I_{CS}(\mathbf{X}; Y) = 0$ if and only if \mathbf{X} and Y are independent of each other and positive otherwise, similarly to the Kullback-Liebler divergence. They also empirically show that the maximum and minimum of Shannon mutual information and quadratic mutual information coincide, but are different in magnitude.

Figure 2.3 depicts the geometrical interpretation of I_{CS} in 2D space (for the case of two discrete random variables), proposed by authors in [7]. We can see that I_{CS} is proportional to the angle between the product of marginal PDFs and the joint PDF.

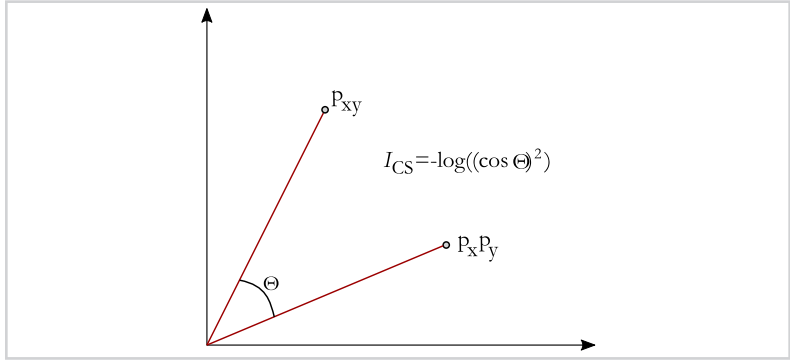


Figure 2.3

Geometrical interpretation of quadratic mutual information.

We can extend the notion of quadratic mutual information to an arbitrary number of variables. Given random variables X_1, \dots, X_D , each with its corresponding PDF p_{X_1}, \dots, p_{X_D} and the joint PDF across all variables $p_{\mathbf{X}}$, the quadratic mutual information between them becomes

$$I_{CS}(X_1; \dots; X_D) = D_{CS}(p_{\mathbf{X}}, \prod_{k=1}^D p_{X_k}). \quad (2.43)$$

Feature selection

A feature is a measurable property of a system being investigated. It can contain raw data obtained from the observations, but often the data are preprocessed or transformed to facilitate further analysis [28]. For example, when doing digit recognition, the images of the digits are usually translated and scaled so that each digit is contained within an area of fixed size. Using a set of features, machine learning algorithms can model a process and predict its future behaviour. This approach involves taking some kind of learning machine (e.g. decision tree, neural network, support vector machine) to train a model using already known input and output data. For example, based on features collected about patients (gender, blood pressure, presence or absence of certain symptoms, etc.) and given the patients' diagnoses – the outputs, we can build a model and use it afterwards as a diagnosis tool for new patients. The input features and the output can be discrete (e.g. gender) or continuous (e.g. body temperature). If the output consists of labels assigned to each instance then we are dealing with a classification problem. If the output is continuous then we have a regression problem on our hands.

Many classification or regression problems involve high-dimensional input data. For example, gene expression data can easily reach into tens of thousands of features [29]. The majority of these features are either irrelevant or redundant for the given classification or regression task. Large number of features can lead to poor inference performance, possible over-fitting of the model, and increased training time [30].

To tackle these problems, dimensionality reduction of the input feature space is an important step in most machine learning tasks. Finding the most relevant information in a possibly overwhelming amount can:

1. improve the generalization performance with respect to the model built using the full feature set,
2. provide a more robust generalization and faster responses of the model to unseen data, and
3. achieve a better understanding and easier interpretation of the data by the researchers.

In general, two approaches are possible to reduce the dimensionality of the input data. Feature extraction transforms the original data into a feature space of fewer dimensions, while retaining the important information in the transformed data. This

transformation may be linear, as in principal component analysis (PCA) [31], but many non-linear transformations also exist [32–34].

Feature selection on the other hand searches through the original feature space and tries to find features that provide the most information about the behaviour of a process we are investigating [35]. Of course, it is possible to use a combination of both approaches when trying to reduce the dimensionality of the observational data. In this thesis we will be dealing with feature selection, which has the benefit of preserving the original data.

3.1 *Feature relevancy*

Feature selections approach tries to find a subset of the original features, which are the most relevant for a given machine learning task. John et al. [36] categorized the relevance of a feature in the following way:

- Strongly relevant features: A feature is strongly relevant for the output, if removing the knowledge about it (from the full set of features) changes the probability distribution of the output values of the model.
- Weakly relevant features: A feature is weakly relevant if it is not strongly relevant and it becomes strongly relevant if we remove a subset of features from the data. Thus, weakly relevant features provide information about the output, but can be replaced by some other features without losing any information about the output – they become redundant.
- Irrelevant features: If the features are neither strongly nor weakly relevant, then they are irrelevant – they don't provide any information about the output.

3.2 *The process of feature selection*

According to Bloom and Langley [35] feature selection methods can be characterized in terms of four basic aspects that determine the search process.

- Search direction: One can start with an empty feature space and then successively add them to the set of selected features. Another possibility is to start with a full set of features and successively remove them. The first approach is usually called forward selection and the latter backward elimination. A combination of

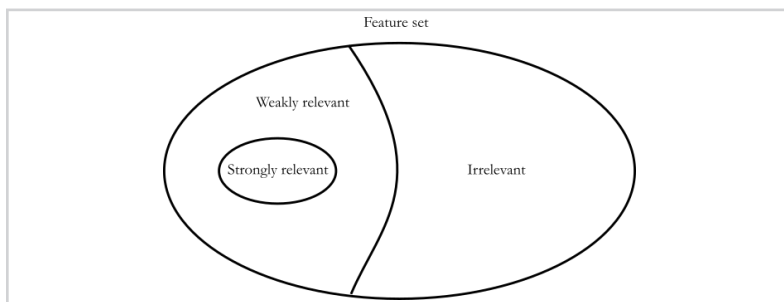


Figure 3.1

Feature categories

the two is also possible, at each step we add and remove features from the set of selected features. This approach tries to avoid hitting a local minima.

- Search strategy: Doing an exhaustive search across the feature space is prohibitive, as there are 2^N possible subsets of N features. Therefore greedy approaches are usually used to pass over the feature space, which can yield sub-optimal solutions, but are much less time consuming.
- Evaluation criterion: Evaluation of the relevance of a feature subset is crucial for the process of feature selection. A more detailed overview of this issue will be given in the next section.
- Stopping criterion: One must decide when to stop the process of feature selection. Multiple approaches are possible. The simplest is to rank the features according to some evaluation criterion and use a parameter to set the breaking point. Usually this parameter is simply the number of features we want, but it can also be more closely connected with the evaluation measure. For example, one could set, as the breaking point, some minimal value for the evaluation criterion a feature must achieve in order to be selected [37]. Another option is to stop adding or removing features when the performance of the model stops improving. Statistical approach is also possible, by first adding irrelevant *probe* features into the original data and then deciding to stop when there is no significant difference in the evaluation criterion value between the probes and the real candidate features [38]. The stopping criterion determines the size of the final feature set, which can affect the model's performance and is arguably dependent

on the number of training instances [39].

3.3 *Feature selection methods*

We can divide feature selection methods into three main groups depending on how they evaluate features; specifically wrapper, embedded, and filter [35].

3.3.1 *Wrapper methods*

The wrapper approach [40] uses the performance of a learning machine to evaluate the relevance of feature subsets. The main idea behind wrappers is that the learning machine, which will use the feature subset, should provide a better assessment than a separate measure based on entirely different foundations.

These methods search through feature subsets and on each of them train some kind of learning machine and use the accuracy of the prediction model as the criterion function for selecting the relevant features. Evaluating 2^N subsets of features is a NP-hard problem, therefore the search process across different subsets is done heuristically, which yields suboptimal subsets, but avoids doing an exhaustive search. Examples of search strategies used are sequential search [41, 42], genetic algorithms [43] or particle swarm optimization [44].

Wrapper methods achieve good performance, but are biased toward a specific learning machine. Their main disadvantage is the computational cost caused by repeated training of the learning machine for each feature subset under consideration. This makes the wrapper approach infeasible for use on large data sets.

3.3.2 *Embedded methods*

Embedded methods reduce the computation time spent on evaluating different subsets which is done in wrapper methods. They integrate feature selection into the learning machine itself and perform feature selection implicitly, during the training phase. Examples of this approach are the recursive partitioning methods for induction found in the decision tree algorithms ID3 [45] and C4.5 [46]. These methods are faster but still dependant on the learning machine.

3.3.3 *Filter methods*

Filters approach to feature selection, in contrast to wrapper and embedded approach, introduces a separate processing step, which occurs before actually training a learning

machine. Filter methods are independent of the learning machine and can be used as a preprocessing step to various machine learning tasks.

The simplest approach is to use a basic relevance criterion based on some measure such as correlation coefficient [47] or mutual information [48] on each individual feature and then select the highest ranked features. More complex variants of this approach incorporate more advanced search strategies and assess feature subsets in order to select the most promising features [49]. Another feature ranking algorithm is Relief [50], but it incorporates a more complex feature evaluation function. On the other hand algorithms like Focus [51] involve a more thorough search through feature space, by first looking at individual features, then pairs, triplets, etc., until an optimal subset is found. The advantages of filter methods are in generally lower computational time and are less prone to over-fitting, but may fail to find the optimal feature subset for a given learning machine. They can overlook inter-feature interactions with regards to the output and select redundant or irrelevant features, due to the limitations of the criterion function. Another drawback is also the fact that choosing the number of features to be selected is usually done manually as no convincing automatic method exist [39].

3.4 *Information-theoretic feature selection*

Evaluation criteria derived from information theory have been widely used in filter methods. Information-theoretic measures such as mutual information are suitable for measuring feature relevancy and redundancy. Mutual information does not assume linear dependencies between features and can handle categorical, and numerical data [52].

Information gain is the simplest of information-theoretic feature selection methods [29]. It computes mutual information between every feature and the output (class) and ranks them accordingly. It is fast, but does not take in consideration dependency between features, so it is unable to detect redundant features. Due to this drawback more complex methods have been proposed in order to select only those features, which are non-redundant with regards to each other.

Battiti [53] proposed a first-order incremental method Mutual information feature selection (MIFS). Given a set of already selected features $\mathbf{X}_S = \{X_1, \dots, X_M\}$, a set of candidate features $\mathbf{X}_C = \{X_{M+1}, \dots, X_N\}$ and the output Y , we compute the criterion

function for each candidate feature $X_c \in \mathbf{X}_C$

$$S_{\text{MIFS}}(X_c) = I(X_c; Y) - \beta \sum_{s=1}^M I(X_c; X_s), \quad (3.1)$$

where $X_s \in \mathbf{X}_S$. At each step of the algorithm we add the candidate feature with the maximum value to the set of already selected features. The criterion is a heuristic which takes into account first order relevancy, assessed by mutual information between the candidate feature and the output $I(X_c; Y)$, and first order redundancy, represented by mutual information between the candidate feature and already selected feature $I(X_c; X_s)$. The method includes a free parameter β , which greatly affects performance [3].

Kwak et al. [3] proposed an improved variation of MIFS called MIFS-U, which avoids some of the deficiencies of the former. Peng et al. [54] further improved on the idea and proposed the minimum redundancy maximum relevance criterion (MRMR), which uses MIFS with automatic setting of parameter β

$$S_{\text{MRMR}}(X_c) = I(X_c; Y) - \frac{1}{M} \sum_{s=1}^M I(X_c; X_s). \quad (3.2)$$

MRMR avoids using parameters, but still considers only first-order interactions. Nevertheless, their modification outperforms MIFS and MIFS-U [55].

Another variation of MIFS, is the normalised mutual information feature selection (NIMFS) [56]. It uses normalized MI in the redundancy term instead of usual MI in order to reduce bias towards multivalued features.

Yang and Moody [57] used joint mutual information (JMI) as a criterion for feature selection

$$S_{\text{JMI}}(X_c) = \sum_{s=1}^M I(X_c, X_s; Y). \quad (3.3)$$

Their criterion considers second order interactions between features and the output, thus increasing computational costs on one hand, but on the other hand also allowing detection of features which, when taken in pairs, provide more information about the output than the sum of both features' individual contributions. Many more MI based feature selection methods exist, but Brown et al. [55] and Vergara et al. [30] showed

that they can be unified in a mutual information feature selection framework, from where each of them can be derived.

There are also a few cases of using mutual information derived from Renyi [58, 59] and Tsallis entropy [60] showing promising results. Chown and Huang [61] proposed using a data compression algorithm along with quadratic mutual information to perform feature selection, but their method is prone to over-fitting, due to the estimation of the criterion in high-dimensional space.

Several methods based on information theory have been developed which go beyond second-order interactions [62–64]. The joint search for multiple features is difficult as a multidimensional probability distributions are hard to estimate, and becomes especially problematic when the number of samples is small. However, this is a favourable approach when questing for a small number of features, as some subtle interactions can be revealed. When using filter methods as a pre-processing stage for a machine learning task, it is usually better to select more features and give the learning machine more options to choose from and possibly find higher-order interactions during the learning phase [65].

These methods are usually used on discrete/discretized data for classification problems with one output (class), however usage of feature selection on multi-class data sets is also possible [66]. Furthermore, Frenay et al. [67] also examine the adequacy of MI for feature selection in regression tasks and argue that in most cases it is a suitable criterion. However, regardless of feature selection being a precursor to classification or regression task, most problems arise from the difficulty of estimating the MI.

Proposed method

Majority of the methods listed in the previous chapter select features according to a criterion made out of two parts: the relevancy term, which rewards features relevant to the output variable, and the redundancy term, which penalizes redundant features. They mostly try to detect first order interactions between features and will fail to select optimal features in case of higher order interactions. Given the classical binary XOR problem in Table 4.1, MIFS and its derivatives fail to notice that features F_1 and F_2 fully define the output and will treat them as irrelevant. This shortcoming is solved in the case of JMI, which would correctly identify both of them, since it considers second order interactions. It should be noted however that one of the features should already be in the set of selected features, this is due to the limitations of the greedy search strategy. Of course, if we define the output as a XOR function of three binary features F_1 , F_2 , and F_3 , then JMI would also completely fail, since it only deals with second order interactions.

Table 4.1
XOR between features F_1 and F_2

F_1	F_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

As previously mentioned, the most common way of estimating Shannon information-theoretic measures for the task of feature selection is through histograms. There are other approaches but each of them with its own limitations (see Chapter 2.3).

In this chapter we present a novel feature selection method based on quadratic mutual information (QMI) [68]. Our motivation is the straightforward estimation of QMI for discrete features, continuous features, or their combination, which makes it suitable for use without any preprocessing dependent on expert knowledge about the data. Moreover, it avoids using parameters, which are inconvenient for non-experts in the field. It is possible to use it as a precursor to classification as well as regression problems in order to avoid over-fitting and to improve the learning machine performance.

As a balance between computational complexity and order of the method, we decided to consider up to second order interactions between features, similarly to JMI.

4.1 Quadratic mutual information feature selection - QMIFS

The quadratic mutual information (2.42) works as the basis for our feature selection method, because it can be computed directly from the data samples and works for both, discrete and continuous features. Optimally, the method should assess every possible subset of feature candidates and select the subset with maximum QMI. However, evaluating all possible subsets of features is prohibitively time consuming. Another problem is that the estimation of I_{CS} is prone to over-fitting, especially if the number of samples is not much larger than the number of features in the subset. This is a common problem in machine learning when dealing with high-dimensional data. To cope with it, feature selection methods usually rank or select features iteratively one by one. Authors in [69] show that complex search techniques often don't provide significant benefits over simple methods like sequential forward selection [42]. With this in perspective we adopt sequential forward selection as the search strategy for our method.

Even if the features are added to the relevant set one by one, it is still important to consider possible interactions between them to prevent adding redundant features, or to include those that are not informative about the output on their own, but are useful when taken with other features.

The proposed method (Algorithm 1) selects features iteratively, until it reaches a stopping criterion – the number of features we want to have. At each step, the algorithm considers all possible candidates from the set of candidate features \mathbf{X}_C . It checks each candidate feature X_c against the set of already selected features $X_s \in \mathbf{X}_S$ from the previous steps using the following heuristic as a criterion function

$$\begin{aligned} S_{\text{QMIFS}}(X_c) &= S_{\text{QMIFS}}(X_c, \mathbf{X}_S, Z) \\ &= \begin{cases} I_{CS}(X_c; Z) & \text{if } M = 0 \\ \sum_{s=1}^M (I_{CS}(X_c; X_s; Z) - I_{CS}(X_c; X_s)) & \text{if } M > 0 \end{cases} \quad (4.1) \end{aligned}$$

where M is the number of already selected features.

It adds the candidate feature X_c with maximum S_{QMIFS} to the set of already selected features. In the beginning \mathbf{X}_S is empty, so the algorithm considers only quadratic

Algorithm 1: Quadratic mutual information feature selection – QMIFS

Data: Set of candidate features \mathbf{X}_C and output Z

Result: Set of selected features indices \mathbf{S}

Standardize \mathbf{X}_C and Z

$\mathbf{X}_S \leftarrow \emptyset$

$\mathbf{S} \leftarrow \emptyset$

while stopping condition not met do

$S_{\max} = 0$

for $X_c \in \mathbf{X}_C$ *do*

$S_c \leftarrow S_{\text{QMIFS}}(X_c, \mathbf{X}_S, Z)$

if $S_c > S_{\max}$ *then*

$S_{\max} \leftarrow S_c$

$X_{\max} \leftarrow X_c$

$c_{\max} \leftarrow c$

end

end

$\mathbf{X}_C \leftarrow \mathbf{X}_C / X_{\max}$

$\mathbf{X}_S \leftarrow \mathbf{X}_S \cup X_{\max}$

$\mathbf{S} \leftarrow \mathbf{S} \cup c_{\max}$

end

mutual information between candidates and output. For later steps the criterion function (4.1) is composed of sums of pairs of terms. The first term rewards the candidate features that are the most informative about the output when taken along with an already selected feature. The second term penalizes the features that have a strong correlation with already selected features. This on one hand ensures detection of features which work better in pairs – they provide more information about the output when taken together than the sum of both features’ individual contributions. On the other hand, it avoids selecting redundant features – the information they provide about the output is present in one of the already selected features. Extension of the criterion to include higher order interactions between features is possible, but considerably increases the computational time and is more prone to over-fitting. Figures 4.1 to 4.7 show the important steps of the algorithm in the case of selecting two features from a set of three candidates.

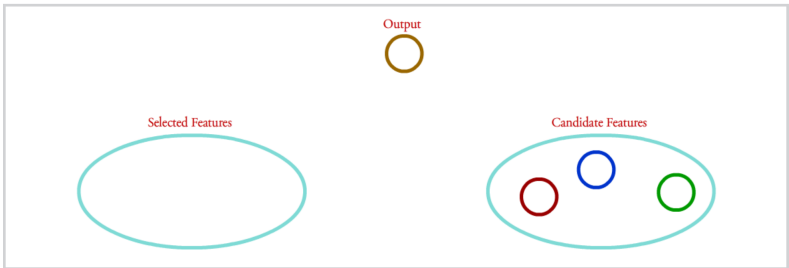


Figure 4.1
In the first step of the algorithm, all the candidate features, along with the output are standardized.

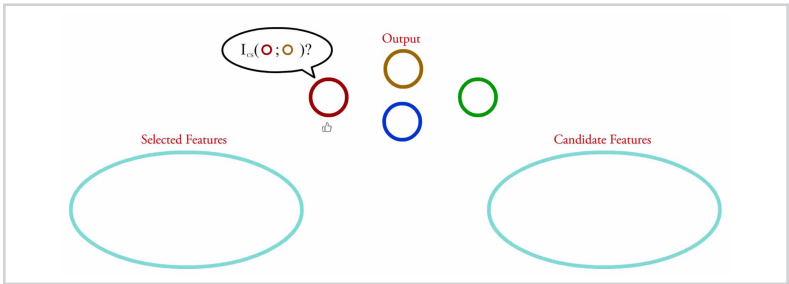


Figure 4.2
Next, we compute I_{CS} for each of the candidate features with regards to the output. The thumbs-up represent the amount of information about the output present in each feature.

Figure 4.3

The feature with the highest I_{CS} is selected first. In our case this is the 'blue circle' feature. The remaining two go back to the set of candidate features.

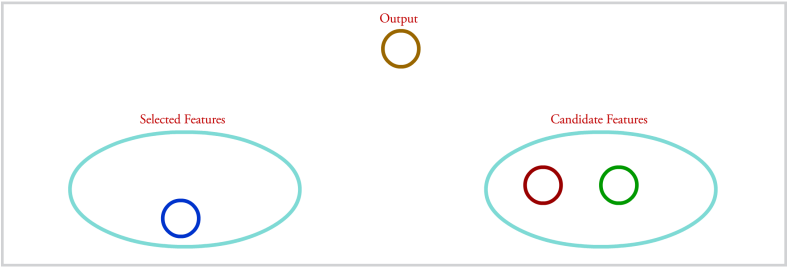


Figure 4.4

Then, for the two remaining candidate features we compute the relevancy term $-I_{CS}$ between the current candidate, the already selected feature, and the output.

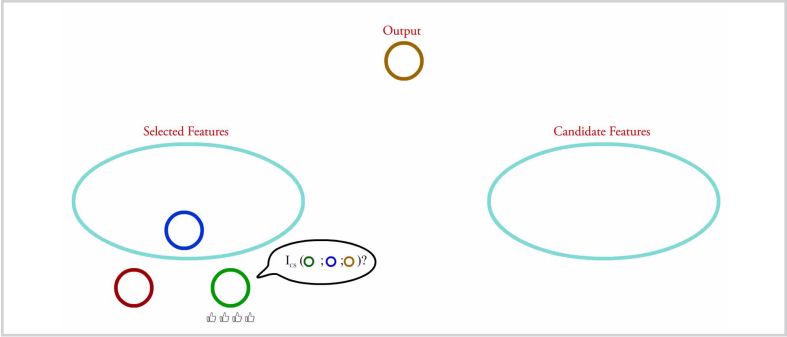
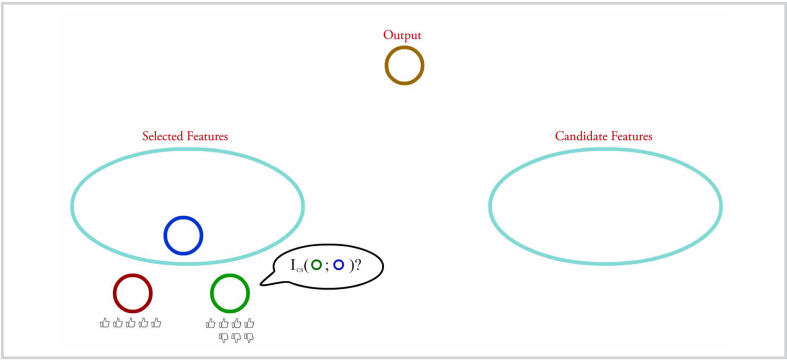


Figure 4.5

Next, we compute the redundancy term, namely I_{CS} between the candidate feature and the already selected feature (thumbs-down) and subtract it from the relevancy term.



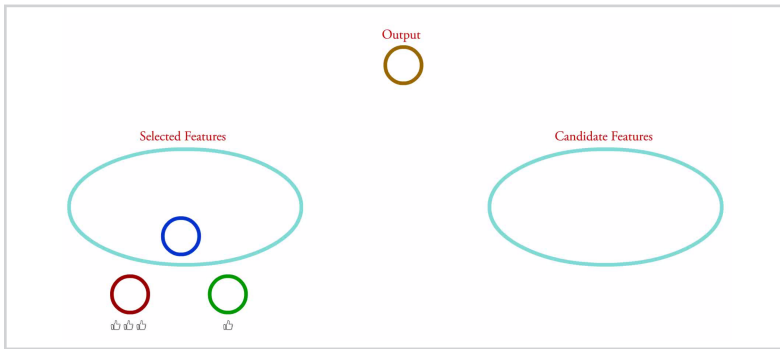


Figure 4.6

The difference between the relevancy term and the redundancy term yields the final score. The candidate feature with the highest score gets selected next. In the case of more features the process repeats again, beginning from Fig. 4.4.

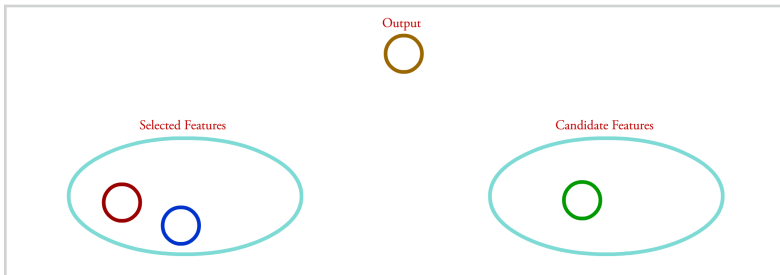


Figure 4.7

The final result is a set of two selected features with the remaining candidate feature discarded.

It would be also possible to generalize the criterion function with the addition of weights for the relevancy and redundancy part of the heuristic. This would give additional control over what we want to emphasize when selecting features (redundancy or relevancy). However sweeping through the weight space showed no general benefit as it is heavily dependent on the data set under consideration. Thus to keep the method simple and parameterless we avoid this generalization. Furthermore we also considered different options for the heuristic, such as using max operation instead of summation already selected features, or including interactions between all the candidate features as the criterion of selecting the first feature. These changes again offer no improvement on real data sets and were abandoned.

Note that the (Algorithm 1) does not define a specific ending condition. In our experiments we fix the number of features we want in order to simplify analysis of performance in comparison to other feature selection methods. However, more so-

phisticated approach could be employed such as using a learning machine on the selected features to see if the addition of the next feature improves the performance of the learning machine. Of course cross-validation has to be performed at each step in order to avoid over-fitting [2, 69].

There are a few considerations we have to take into account before using our method for feature selection. Firstly, the estimation of I_{CS} depends heavily on the kernel width σ [7]. The Silverman rule [23] is a common way to estimate it, but the width must be the same across all features. Neglecting this, the value of the criterion function will vary even if all candidate features are equally relevant to the output [7], and will fail to choose the correct ones. We take care of this problem by standardizing (rescaling to zero mean and standard deviation of one) the data, which in turn causes the Silverman rule to produce the same σ for every feature. Secondly, the magnitude of I_{CS} has no meaning [7] due to the dependence on the choice of window width. However, correct identification of the most relevant features requires only relative difference among them. That is, given two features X_a , X_b , and the output, and knowing that feature X_a is more informative about the output than X_b , the S_{QMIFS} estimate is acceptable as $S_{QMIFS}(X_a) > S_{QMIFS}(X_b)$. The following small-scale experiment nicely presents some of the important properties of the proposed criterion.

We generate correlated data composed from two features X_s , X_c and an output Z . All three are continuous with 2000 samples drawn from normal distribution with zero mean and unit variance. We assume that feature X_s is already in the set of selected features, and treat X_c as the current candidate. Figure 4.8a shows how $S_{QMIFS}(X_c)$ changes while keeping correlation $\text{corr}(X_c, X_s)$ fixed at 0.1, $\text{corr}(X_s, Z)$ at 0.6, and varying the correlation between X_c and output Z from 0 to 1. As the correlation increases, $S_{QMIFS}(X_c)$ also increases, but non-linearly. This behaviour is expected since correlation is not comparable to quadratic mutual information. Figure 4.8b shows the opposite, how increasing the correlation between features affects the criterion value. We fix correlations $\text{corr}(X_c, Z)$ and $\text{corr}(X_s, Z)$ to 0.6 and vary the $\text{corr}(X_c, X_s)$ from 0 to 1. The result shows that S_{QMIFS} penalizes redundant features – the higher the redundancy (represented here as inter-feature correlation) the lower the criterion value. These findings demonstrate that S_{QMIFS} follows the aforementioned propriety of guaranteeing the correct ordering of features.

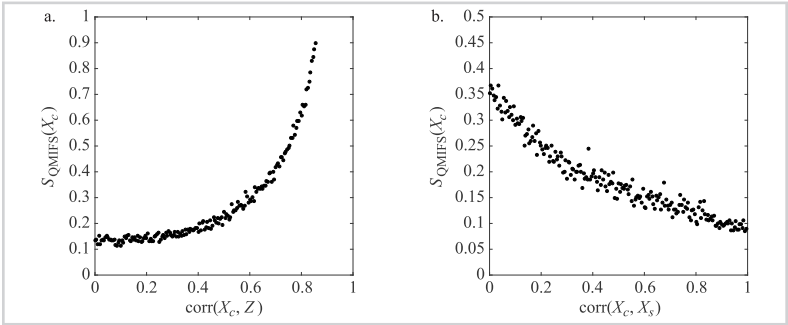


Figure 4.8
Properties of quadratic mutual information feature selection criterion: a. relevance of feature X_c as its correlation with the output Z increases, and b. redundancy of feature X_c as its inter-feature correlation with already selected feature X_s increases.

4.2 Performance on artificial data sets

In order to illustrate the capabilities and shortcomings of our method we perform feature selection on three artificial data sets with known concepts behind the output – we know which features affect in advance.

Table 4.2

Summary of artificial data sets

Data set	Instances	Features	Output concept
xor2+5	128	7	$Y = X_3 \oplus X_5$
xor5+5	1024	10	$Y = X_2 \oplus X_3 \oplus X_4 \oplus X_6 \oplus X_8$
corral	128	7	$Y = (X_1 \wedge X_2) \vee (X_3 \wedge X_4)$

Data set xor2+5 consists of 7 binary features and a binary output with 128 instances. The output is simply defined as a binary xor function of features X_3 and X_5 with X_3 being slightly (10%) correlated with the output. The rest of the features are irrelevant. In this case QMIFS and JMI are able to recognize the two relevant features since the criterion function takes into account second order interactions between candidate features. Methods like MIFS and MRMR fail in this case as they only consider first order interactions. Both QMIFS and JMI use sequential forward selection and would succeed only by chance to find the two relevant features if one of them was not slightly correlated with the output. We can avoid this problem by examining all possible pairs at the beginning of the search process and select the one that interacts the most with the others as the first one. This introduces significant computational burden, which

is not desirable for a filter method. Moreover, preliminary experiments performed on real data show that it does not improve performance. This means that such relations are rare in practice.

The second data set `xor5+5` is an extension of the first one by adding three additional binary features. Now the output is defined as an xor between features X_2 , X_3 , X_4 , X_6 , and X_8 . Five irrelevant features remain in the data set. We expand the number of instances to 1024. Our method, as well as MIFS, MRMR, and JMI fail in this case as they don't consider such high order of interactions between features. This is obvious a shortcoming of our method, but including higher order interactions into the criterion function, although possible, would prohibitively increase computational time.

In the third case we use the `corral` data set [36]. There are 7 binary features and 128 instances in this data set. The output Y for each instance is defined as

$$Y = (X_1 \wedge X_2) \vee (X_3 \wedge X_4) , \quad (4.2)$$

where X_1 , X_2 , X_3 , and X_4 are the first four features. Feature X_5 is irrelevant, and feature X_6 is highly correlated with the output but has a 25% error rate. This data sets exposes the shortcomings of forward selection. Due to the high correlation between feature X_6 and the output QMIFS, similarly as the MIFS, MRMR, and JMI selects this feature first and only then adds the other four relevant features, which fully describe the output. This causes problems to some machine learning algorithms as feature X_6 only contributes noise and is in essence redundant. Methods that include backward elimination in the process of feature selection cope better with this kind of problem as they remove features from the candidate feature set rather than choosing them.

4.3 *Results and discussion*

For our experiments, we use 16 data sets; most are from the UCI machine learning repository [70], one is from a company which deals with web advertisement placement, others are from various papers or projects. To compare the methods over a wide variety of scenarios we choose the data sets so that some include only discrete data, some only continuous and some mixed. The experiments cover two problem domains: one is dealing with classification and the other with regression. Table 4.3 briefly summarizes the information about the data sets. For each data set it lists number of instances, number of features and their type, the type of the output, and the problem domain.

Table 4.3

Main properties of used data sets.

<i>Data set</i>	<i>Instances</i>	<i>Features</i>		<i>Output</i>	<i>Problem domain</i>
		<i>discrete</i>	<i>continuous</i>		
Chess	1000	36	0	binary	Classification
Breast cancer	569	0	30	binary	
Ionosphere	351	0	34	binary	
Sonar	208	0	60	binary	
Wine	178	0	13	ternary	
Cervical cancer	58	0	714	binary	
Spine	310	0	12	binary	
Leukemia	73	0	1868	binary	
Communities	1993	0	100	continous	Regression
Parkinson telemonitoring	1000	0	16	continous	
Wine quality	1599	0	11	continous	
Housing	506	12	1	continous	
Web advertisement	950	38	8	continous	
Facebook	10000	0	53	continous	
Blog	5000	0	280	continous	
Bank	4500	0	32	continous	

Short descriptions of selected data sets:

- *Chess* – data set describes the end game scenario in chess of king and rook (white) versus king and pawn (black). The 36 features describe the positions of the figures on the board for each case. The output has two values either "white can win" (52% of instances) or "white cannot win" (48%).
- *Breast cancer* – the features are computed from a digitized image of a cell nuclei sample and are all real valued. The output can obtain two values either the sample is benign (63% of cases) or malignant (37%).
- *Ionosphere* – the data are from a radar system used to probe the ionosphere for evidence of some type of structure. The radar returns are described using 34 features and the output is two valued, either the radar signal passed through the ionosphere (35.9%) or there was some echo back to the radar station (63.8%).
- *Sonar* – the data were obtained by bouncing sonar signals from metal cylinders or

rocks under various angles. The output is again two labelled either representing a rock (46.6%) or a metal cylinder (53.4%). The 60 features represent power of the bounced signal for different sonar frequency bands.

- *Wine* – the data are the results of a chemical analysis of wines grown in the same region of Italy, but derived from three different cultivators. The 13 features represent the quantities of constituents found in each of the three types of wine. The output represents the cultivator to which a specific sample of wine belongs to. The distribution of samples with regards to cultivators is (33.1% , 39.9% , 27.0%).
- *Cervical cancer* – consists of gene expression profiling data from tumor and matched control samples, 50% each. The data are raw read counts from sequencing of microRNA. It was published by Witten et al. [71].
- *Spine* – the data set contains physical properties patients' spines with the output being if a specific patient shows abnormalities or not regarding the condition of the spine. The distribution of the output is 67% for the abnormal cases and 33% for normal.
- *Leukemia* – consists of gene expression data obtained from bone marrow samples of patients. The output is the diagnosis of the patients involved – diagnosed with leukaemia (33%) or not (66%). A study using this data was published in [72].
- *Communities* – the data combine socio-economic data, law enforcement data, and crime data for different cities in the US. The output is the total number of violent crimes per 100k population.
- *Parkinson telemonitoring* – combines a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease during a six month period. The output is the clinician's Parkinson's disease symptom score.
- *Wine quality* – the data combine physiochemical proprieties of red and white wine variants of a Portuguese wine. The output is the quality of the wine (a score from 1 to 10).
- *Housing* – this data set contains information collected by the US Census service concerning housing in the area of Boston Mass. The output is the median value of owner-occupied homes.

- *Web advertisement* – combines a range of features with regards to the device, location, time, etc. for a displayed advertisement. The output of each instance is the measured load time of an advertisement on a web page or mobile application.
- *Facebook* – instances in this data set contain features extracted from Facebook posts. The output gives the number of comments a post will receive in a specified time frame. A study on analysis of this data set was published in [73].
- *Blog* – this data set originates from blog posts [74]. Features were extracted from raw HTML data. The output is the number of comments a blog post will receive in the next 24 hours.
- *Bank* – a synthetically generated data set from simulation of how bank-customers choose their banks. The output gives the rate of rejections for each bank i.e. the fraction of customers that are turned away from the bank because of long queue times. The data set is from the Delve project [75].

4.3.1 *Experimental methodology*

We compare our method QMIFS to three other common and comparable methods which use information-theoretic approach to feature selection: MIFS with $\beta = 1$, MRMR, and JMI. Among the multitude of information-theoretic methods, we chose MIFS because of its historical importance and widespread usage in relevant literature. MRMR was selected because it is among the best improvements of MIFS [76]. The third method JMI, was chosen as a representative of second-order methods due to its promising performance [76]. All of them use the same search strategy as QMIFS – sequential forward selection and need discretization of the continuous features before using them.

Additionally we include ReliefF [77] feature ranking method in the comparison, since it is very common in literature, but is not based on information-theoretic measures. It selects features by considering distances between feature vectors with regards to the output. It does not use a particular search strategy, as it only assigns scores to features, and then the user decides on the number of top ranking features. We need to provide the method with one parameter: the number of neighbouring instances it checks when computing the scores. If this parameter equals 1, the estimates computed by ReliefF can be unreliable for noisy data. If the value is comparable with the number of instances, ReliefF can fail to find important features. We set this number to

10 – the default value recommended by the Matlab package. It is noise-tolerant and capable of detecting feature interactions, however it does not discriminate between redundant features and has difficulties dealing with low number of samples. In some cases a combination of ReliefF and information-theoretic methods is used to overcome the shortcomings [78].

For methods that need discrete data to work (MIFS, MRMR, JMI) we use minimum description length discretization (MDL) procedure from WEKA [79], which promises better results than the usual approach of equal frequency or equal width binning [18]. We use it on every data set that includes continuous features in order to make them discrete. QMIFS and ReliefF can handle both discrete and continuous data so no preprocessing is needed.

Classification tree from the Matlab Statistics and Machine Learning Toolbox serves as the indirect performance evaluation tool on the classification problem domain. We choose this method because it is simple, parameterless, deterministic, fast and can benefit from interactions present in the selected features, contrary to some other methods like Naive Bayes, which assume that there are no dependencies amongst attributes [80]. More complex methods like gradient boosting trees [81] were also considered, but preliminary tests showed no fundamental differences in the performance of feature selection methods.

We evaluate the performance of the methods using three measures:

1. Classification accuracy (CA), defined as the number of correctly classified samples divided by the total number of samples.
2. Area under the curve (AUC), which is the probability that a classifier will rank a randomly chosen positive sample higher than a randomly chosen negative one (assuming positive is higher than negative). A perfect classifier has this score equal to 1, random classification has a score of 0.5.
3. Youden index (Y-index) is the difference between true positive rate and false positive rate. It is defined for all points on the receiver operating characteristic curve (ROC), but in our case we provide the value in the optimal point for the given classifier. It's value ranges from -1 to 1 . A value of 1 indicates that there are no false positives or false negatives – perfect score.

In the regression problem domain we assess the performance using the regression tree

from the Matlab Statistics and Machine Learning Toolbox because of the same reasons as in the case of classification. We obtain the performance of the methods through measuring the root-mean-square-error (RMSE)

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (\hat{y}_i - y_i)^2}{n}}, \quad (4.3)$$

where n is the number of samples and \hat{y}_i and y_i are the estimated value and the true value of the output.

As the output is continuous, MDL discretization is useless. Instead, equal frequency binning is used, with five bins for every feature and output. Equal frequency binning usually works better than equal width binning [18], and the empirical evidence from experimenting with MDL discretization shows that the number of bins per feature is often between three and seven. Therefore for every continuous feature in each data set we split the values into five groups so that the amount of values is approximately the same across all groups and substitute each value in the data with the index of the group it belongs to.

In both problem domains one thousand hold-out validations are performed on each data set. Each time two thirds of randomly sampled instances act as the training set to build the model and the rest as the validation set to measure the performance. For each method we vary the number of selected features: 3, 5, 7, or 10, and compare the results against the baseline performance where all features are used to train the model.

To get a clearer representation of result in both problem domains we rank the methods according to the measures CA, AUC, Y-index, and RMSE. Each method obtains a rank from 1 (best) to 5 (worst). The ranked values get the same (average) rank if their 95% confidence intervals overlap.

4.3.2 Classification performance

Table 4.4 shows which features are selected by each method, and Tables 4.5 and 4.6 summarize the rankings of the methods for each test scenario for measures CA, AUC, and Y-index and their average ranks. The ranks imply that all three measures behave similarly, which is expected since the data sets are reasonably well balanced with respect to the number of class values. Tables 4.7 and 4.8 show a more detailed insight into the performance of the methods for seven selected features. It includes only the maximum

standard error (SE) of the performance indexes since standard errors across different methods are practically the same. Standard error is defined as

$$SE = \frac{s}{\sqrt{R}}, \quad (4.4)$$

where s is the sample standard deviation of the measurements and R the number of measurements done. Results for three, five, and ten selected features are in the Appendix [A](#).

Table 4.4

Classification problem domain: selected features.

Data set	Method	Selected features			
		3	5	7	10
Chess	MIFS	21, 10, 33	32, 9	1, 2	3, 4, 5
	MRMR	21, 10, 33	32, 15	8, 16	18, 6, 27
	JMI	21, 10, 33	32, 15	8, 16	18, 6, 7
	QMIFS	33, 10, 21	35, 6	8, 18	7, 15, 13
	ReliefF	33, 10, 21	27, 35	16, 32	15, 2, 23
Breast cancer	MIFS	23, 22, 5	19, 10	12, 15	30, 18, 29
	MRMR	23, 22, 28	14, 27	21, 8	29, 25, 24
	JMI	23, 28, 24	8, 22	21, 4	7, 27, 14
	QMIFS	28, 2, 23	8, 27	21, 22	3, 7, 1
	ReliefF	22, 25, 2	21, 27	23, 28	29, 8, 7
Ionosphere	MIFS	5, 16, 2	18, 1	10, 30	32, 7, 24
	MRMR	5, 16, 18	27, 3	6, 7	4, 34, 31
	JMI	5, 6, 3	33, 8	17, 21	4, 13, 29
	QMIFS	5, 6, 33	15, 8	7, 21	28, 31, 24
	ReliefF	8, 6, 24	16, 14	4, 34	5, 3, 12
Sonar	MIFS	11, 51, 36	44, 4	1, 2	3, 6, 7
	MRMR	11, 51, 36	48, 12	9, 54	45, 4, 21
	JMI	11, 4, 12	48, 9	21, 45	10, 36, 49
	QMIFS	12, 27, 11	48, 10	16, 9	13, 49, 28
	ReliefF	10, 9, 12	11, 26	8, 37	31, 27, 36
Wine	MIFS	7, 1, 11	5, 3	4, 9	8, 10, 2
	MRMR	7, 1, 13	11, 10	12, 6	5, 2, 4
	JMI	7, 1, 13	11, 10	12, 6	2, 5, 4
	QMIFS	7, 1, 13	12, 10	11, 6	5, 9, 4
	ReliefF	13, 1, 4	5, 7	9, 3	10, 11, 12
Cervical cancer	MIFS	34, 562, 19	632, 695	1, 2	3, 4, 5
	MRMR	34, 562, 73	188, 189	72, 427	102, 399, 632
	JMI	34, 180, 188	73, 562	273, 618	72, 427, 92
	QMIFS	165, 644, 72	327, 657	398, 73	172, 224, 186
	ReliefF	452, 72, 73	382, 642	355, 539	180, 684, 413
Spine	MIFS	6, 5, 2	7, 8	9, 10	11, 12, 4
	MRMR	6, 5, 2	4, 7	3, 8	9, 1, 10
	JMI	6, 5, 1	2, 3	4, 7	8, 9, 10
	QMIFS	6, 4, 5	3, 1	2, 9	10, 11, 12
	ReliefF	5, 6, 4	1, 2	3, 11	9, 8, 10
Leukemia	MIFS	817, 1238, 915	637, 1	2, 3	4, 5, 6
	MRMR	817, 1238, 1671	493, 454	596, 1566	190, 800, 431
	JMI	817, 1238, 493	454, 1671	1566, 596	431, 800, 191
	QMIFS	672, 627, 431	1828, 1114	606, 190	275, 1106, 1238
	ReliefF	454, 622, 817	493, 596	1620, 1238	1621, 1566, 1296

Table 4.5

Classification problem domain: ranking of feature selection methods for the first half of the data sets. Ranks calculated from the measures CA, AUC, and Y-index are presented as triplets CA/AUC/Y-index.

Data set	Method	Selected features				Average
		3	5	7	10	
Chess	MIFS	3/3/3	3/3/3	2.5/5/4	4/5/5	3.1/4/3.8
	MRMR	3/3/3	3/3/3	5/3.5/4	2.5/1.5/2.5	3.4/2.8/3.1
	JMI	3/3/3	3/3/3	4/3.5/4	2.5/3/2.5	3.1/3.1/3.1
	QMIFS	3/3/3	1/1/1	1/1/1	1/1.5/1	1.5/1.6/1.5
	ReliefF	3/3/3	5/5/5	2.5/2/2	4.5/4/4	3.8/3.5/3.5
Breast cancer	MIFS	3.5/4/3.5	2.5/3/3	5/3/5	2/4.5/3	3.3/3.6/3.6
	MRMR	1.5/2/1.5	2.5/3/2	2.5/3/1.5	4.5/2/3	2.8/2.5/2
	JMI	3.5/2/3.5	1/1/1	2.5/3/1.5	2/2/2.5	2.3/2/2.3
	QMIFS	1.5/2/1.5	4.5/3/4.5	2.5/3/3.5	2/2/3	2.6/2.5/3.1
	ReliefF	5/5/5	4.5/5/4.5	2.5/3/3.5	4.5/4.5/3	4.1/4.4/4
Iono-sphere	MIFS	3.5/3.5/3.5	1.5/3.5/2.5	1.5/3/1.5	3/3/3	2.4/3.3/3.5
	MRMR	3.5/3.5/3.5	3.5/2/2.5	3.5/3/4	4/4.5/4	3.6/3.3/3.5
	JMI	1/1/1	1.5/1/1	1.5/1/1.5	1.5/1.5/1.5	1.4/1.1/1.3
	QMIFS	2/2/2	3.5/3.5/4	5/5/5	5/4.5/4	3.9/3.8/4
	ReliefF	5/5/5	5/5/5	3.5/3/3	1.5/1.5/1.5	3.8/3.6/3.6
Sonar	MIFS	3/3/3	1/1/1	2/2/2	3.5/3.5/3.5	2.4/2.4/2.4
	MRMR	4/4/3	3.5/4/4	4/4/4	1.5/1.5/1.5	3.3/3.4/3.1
	JMI	1/1.5/1	2/2.5/2	2/2/2	1.5/1.5/1.5	1.6/1.9/1.6
	QMIFS	2/1.5/3	3.5/2.5/3	2/2/2	3.5/3.5/3.5	2.8/2.4/2.9
	ReliefF	5/5/5	5/5/5	5/5/5	5/5/5	5/5/5

Table 4.6

Classification problem domain: ranking of feature selection methods for the second half of the data sets. Ranks calculated from the measures CA, AUC, and Y-index are presented as triplets CA/AUC/Y-index. The row with the overall averages includes also the data from Table 4.5.

Data set	Method	Selected features				Average
		3	5	7	10	
Wine	MIFS	1/2.5/1	3/3/3	3/2.5/3	1.5/1.5/2.5	2.1/2.4/2.4
	MRMR	3/2.5/3	3/3/3	3/2.5/3	4/3/4	3.3/2.8/3.3
	JMI	3/2.5/3	3/3/3	3/2.5/3	4/3/2.5	3.3/2.8/2.9
	QMIFS	3/2.5/3	3/1/1	3/2.5/1	1.5/1.5/1	2.6/1.9/1.5
	ReliefF	5/5/5	3/5/5	3/5/5	4/5/5	3.8/5/5
Cervical cancer	MIFS	2.5/2.5/2.5	3/2.5/3	3/3/3	2.5/2/2.5	2.8/2.5/2.8
	MRMR	2.5/2.5/2.5	1.5/2.5/1.5	1.5/1.5/1.5	2.5/3/2.5	2/2.4/2
	JMI	1/1/1	1.5/1/1.5	1.5/1.5/1.5	1/1/1	1.2/1.1/1.3
	QMIFS	4/4/4	4.5/4.5/4.5	4/4/4	4/4.5/4	4.1/4.3/4.1
	ReliefF	5/5/5	4.5/4.5/4.5	5/5/5	5/4.5/5	4.9/4.8/4.9
Spine	MIFS	3/3/3.5	5/5/5	5/5/5	3/3/3	4/4/4.1
	MRMR	3/3/3.5	2.5/2.5/2.5	2.5/2.5/2.5	3/3/3	2.8/2.8/2.9
	JMI	5/5/3.5	2.5/2.5/2.5	2.5/2.5/2.5	3/3/3	3.3/3.3/2.9
	QMIFS	1/1/1	2.5/2.5/2.5	2.5/2.5/2.5	3/3/3	2.3/2.3/2.3
	ReliefF	3/3/3.5	2.5/2.5/2.5	2.5/2.5/2.5	3/3/3	2.8/2.8/2.9
Leukemia	MIFS	2.5/2.5/2	3/2.5/1.5	1.5/4/1.5	1/5/1.5	2/3.5/1.6
	MRMR	2.5/2.5/2	3/2.5/4	4/2/4	3.5/2.5/4	3.3/2.4/3.5
	JMI	2.5/2.5/4.5	3/2.5/4	4/2/4	3.5/2.5/4	3.3/2.375/4.1
	QMIFS	2.5/5/2	3/5/1.5	1.5/5/1.5	3.5/2.5/1.5	2.6/4.4/1.6
	ReliefF	5/2.5/4.5	3/2.5/4	4/2/4	3.5/2.5/4	3.9/2.4/4.1
Average	MIFS		2.8/3.2/2.9			
	MRMR		3.0/2.8/2.9			
	JMI		2.4/2.2/2.4			
	QMIFS		2.8/2.9/2.7			
	ReliefF		4/4/4.1			

Chess data set: Baseline performs better in this case – looks like the learning machine can handle all 36 features. CA drops by about 0.03 after reducing the number of features to seven and all the methods show similar behaviour at prioritizing features. According to Table 4.5 and tables in Appendix A our method is better than the others when selecting 5, 7, and 10 features, with regards to all three performance indexes.

Breast cancer data set: In this case classification tree benefits from feature selection,

even with only three features selected. Table 4.5 shows that all methods perform similarly; the largest discrepancy among them being at five selected features, where JMI overcomes the others in all three performance indexes.

Ionosphere data set: All methods improve the performance compared to the baseline. Our method does not perform very well in terms of CA, AUC, and Y-index, even though Table 4.4 shows that 5 out of 7 selected features are the same as in the best performing method – JMI. It ranks second at 3 selected features but then falls behind when selecting more of them. Interestingly ReliefF, although worse at 3 and 5 selected features improves further on and reaches JMI's performance on 10 features.

Sonar data set: Only a few features are common to all the methods, so the performance varies substantially between them. At 3 selected features JMI and QMIFS work the best and offer similar performance, having the same AUC ranks, with CA and Y-index being worse for QMIFS. At 5 features CA improves for all methods but is overall still worse than the baseline. Using 7 features selected by MIFS, JMI, or QMIFS offers a considerable improvement in comparison to the baseline (3% better CA). The methods achieve the same ranks since the differences between them are small, causing the confidence intervals to overlap. At 10 selected features all the methods offer improvement over the baseline with JMI and MRMR having 2% better CA than QMIFS and MIFS. Surprisingly, ReliefF lacks behind other methods in all cases.

Wine data set: Feature selection improves performance in comparison to the baseline even though there are only 11 features in the data set. All methods select similar features, manifesting in similar performance, except for ReliefF, which is somewhat worse than the information-theoretic methods. This can be seen in the rankings and in Table 4.8. QMIFS achieves the best ranks when selecting 5, 7, or 10 features.

Cervical cancer data set: Performance of the Classification tree improves if feature selection is done beforehand for all methods. Interestingly methods based on Shannon mutual information select quite similar features, whereas QMIFS and ReliefF identify other features as relevant. This is also evident from the rankings, with the aforementioned methods lagging behind the other three.

Spine data set: Again, feature selection improves performance, with QMIFS having best performance on 3 selected features (about 3% better than the second best method), with methods achieving similar performance as additional features are selected. This is reasonable due to the fact that there are only 12 present in the data set.

Leukemia data set: Here we can see that the Classification tree fails completely.

Although the CA is very high this is due to the fact that the false positive rate is very high – the tree classifies almost all instances to the same class. This is evident from the other two measures AUC and Y-index, which are very low. According to additional experiments, using more advanced learning machine such as Gradient boosting trees does not improve the results meaningfully. This is true in both cases with or without feature selection. Contributing to this failure is probably the fact that there are very few instances and a lot of features in the data set. The ranks are shown in Table 4.6 but are probably meaningless due to the failure of the learning machine. We can not establish whether the selected features are relevant for the output or not.

Table 4.7

Classification problem domain: values of measures for 7 selected features for the first half of the data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Chess	MIFS	0.934	0.956	0.865
	MRMR	0.932	0.964	0.866
	JMI	0.933	0.965	0.866
	QMIFS	0.938	0.976	0.886
	ReliefF	0.935	0.957	0.869
	All features	0.969	0.985	0.942
	SE	0.0004	0.0003	0.0007
Breast cancer	MIFS	0.929	0.939	0.858
	MRMR	0.932	0.942	0.868
	JMI	0.935	0.944	0.873
	QMIFS	0.932	0.941	0.866
	ReliefF	0.932	0.941	0.863
	All features	0.922	0.926	0.848
	SE	0.0006	0.0008	0.001
Ionosphere	MIFS	0.897	0.904	0.794
	MRMR	0.888	0.903	0.780
	JMI	0.897	0.915	0.796
	QMIFS	0.881	0.897	0.764
	ReliefF	0.891	0.907	0.788
	All features	0.878	0.881	0.758
	SE	0.001	0.001	0.002
Sonar	MIFS	0.728	0.755	0.504
	MRMR	0.681	0.711	0.424
	JMI	0.731	0.759	0.510
	QMIFS	0.734	0.759	0.506
	ReliefF	0.674	0.702	0.407
	All features	0.714	0.724	0.460
	SE	0.002	0.002	0.003

Table 4.8

Classification problem domain: values of measures for 7 selected features for the second half of data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Wine	MIFS	0.911	0.919	0.826
	MRMR	0.910	0.917	0.828
	JMI	0.911	0.916	0.829
	QMIFS	0.915	0.923	0.839
	ReliefF	0.913	0.910	0.813
	All features	0.906	0.912	0.813
	SE	0.0014	0.0017	0.0030
Cervical cancer	MIFS	0.828	0.858	0.669
	MRMR	0.848	0.862	0.708
	JMI	0.851	0.879	0.716
	QMIFS	0.803	0.824	0.638
	ReliefF	0.753	0.772	0.525
	All features	0.761	0.785	0.536
	SE	0.003	0.003	0.007
Spine	MIFS	0.766	0.769	0.527
	MRMR	0.797	0.821	0.589
	JMI	0.797	0.831	0.586
	QMIFS	0.798	0.831	0.589
	ReliefF	0.804	0.834	0.597
	All features	0.788	0.791	0.566
	SE	0.001	0.002	0.003
Leukemia	MIFS	0.928	0.079	0.003
	MRMR	0.915	0.091	0.000
	JMI	0.925	0.065	0.003
	QMIFS	0.916	0.093	0.000
	ReliefF	0.913	0.095	0.000
	All features	0.865	0.147	0.000
	SE	0.003	0.002	0.001

Tables 4.7 and 4.8 reveal that CA, AUC, and Y-index behave similarly, except in the case of Leukemia data set, due to the fact that the data sets used are well balanced

in terms of class values. In all cases except the Chess data set, the classification tree benefits from the feature selection with a 0.01 – 0.03 increase in CA. The differences between methods in terms of CA, AUC, and Y-index are small, relative difference is mainly less than 1%.

Overall, QMIFS offers performance similar to the other methods in terms of CA, AUC, and Y-index. Its average ranks shown in Table 4.6 across all data sets and number of selected features are 2.8/2.9/2.7, placing it somewhere in the middle – better than MIFS, MRMR, and ReliefF, but lagging behind JMI. The subtle differences in the rankings can be attributed to the fact that both QMIFS and JMI are second-order methods and can detect some more peculiar relations between features. The difference between QMIFS and JMI could be attributed to the superiority of MDL discretization compared to the direct estimation in the case of QMIFS. The overall worse performance of ReliefF could be connected to the fact that the parameter is not fine tuned for each data set.

To establish if these small differences in rankings are statistically significant we use the Friedman statistical test [82, 83], which enables pairwise comparison of multiple methods on multiple data sets. The test shows no significant differences, probably due to the fact that many more data sets would be needed to show whether the differences observed above are really significant.

4.3.3 Regression performance

Table 4.9 shows which features are selected by each method. Tables 4.10 and 4.11 summarize the ranking of the methods for each test scenario and the average ranks. Tables 4.12 and 4.13 present a more detailed insight into the performance of the methods with regards to RMSE. They only show the results for seven selected features. Experimental results for three, five, and ten features are in the Appendix A.

Table 4.9

Regression problem domain: selected features.

Data set	Method	Selected features			
		3	5	7	10
Communities	MIFS	45, 52, 67	95, 97	48, 36	24, 15, 89
	MRMR	45, 52, 4	41, 51	72, 69	18, 3, 50
	JMI	45, 4, 44	50, 69	51, 46	41, 3, 16
	QMIFS	45, 41, 78	42, 44	4, 68	29, 39, 16
	ReliefF	3, 75, 100	51, 4	78, 72	71, 18, 91
Parkinson telemonitoring	MIFS	15, 12, 14	16, 8	5, 10	2, 13, 4
	MRMR	15, 12, 14	8, 16	2, 10	5, 13, 7
	JMI	15, 14, 2	6, 13	9, 4	10, 7, 11
	QMIFS	15, 14, 13	2, 10	8, 4	11, 9, 6
	ReliefF	15, 2, 14	12, 7	4, 10	6, 9, 11
Wine quality	MIFS	11, 10, 6	4, 9	2, 5	8, 7, 3
	MRMR	11, 10, 2	5, 7	8, 4	9, 3, 6
	JMI	11, 10, 8	3, 2	5, 7	1, 4, 6
	QMIFS	11, 10, 2	8, 3	7, 1	9, 6, 5
	ReliefF	11, 2, 3	4, 10	8, 1	6, 5, 9
Housing	MIFS	13, 11, 4	6, 12	7, 9	8, 2, 10
	MRMR	13, 11, 6	12, 7	10, 4	3, 1, 5
	JMI	13, 6, 11	3, 1	10, 5	2, 7, 9
	QMIFS	13, 8, 3	6, 7	11, 5	10, 9, 4
	ReliefF	6, 5, 13	8, 7	11, 10	12, 2, 9
Web advertisement	MIFS	29, 9, 21	44, 35	15, 13	12, 22, 6
	MRMR	29, 9, 21	44, 35	39, 13	12, 15, 22
	JMI	29, 4, 41	30, 28	10, 39	31, 3, 37
	QMIFS	3, 8, 40	2, 4	46, 16	36, 10, 45
	ReliefF	31, 32, 29	39, 30	42, 22	41, 43, 18
Facebook	MIFS	39, 31, 50	38, 43	36, 35	40, 37, 53
	MRMR	39, 31, 35	23, 5	37, 34	15, 2, 50
	JMI	39, 31, 15	34, 10	20, 32	5, 33, 28
	QMIFS	35, 4, 30	17, 11	32, 16	27, 6, 33
	ReliefF	31, 33, 30	34, 35	25, 16	14, 26, 11
Blog	MIFS	227, 52, 61	37, 62	21, 54	270, 48, 55
	MRMR	227, 52, 37	114, 53	10, 213	265, 61, 232
	JMI	227, 52, 193	30, 189	3, 25	120, 55, 35
	QMIFS	61, 23, 44	51, 21	9, 49	54, 53, 7
	ReliefF	61, 52, 266	248, 54	69, 270	264, 191, 233
Bank	MIFS	12, 6, 18	32, 23	19, 28	3, 1, 4
	MRMR	12, 6, 18	32, 23	19, 28	14, 10, 27
	JMI	12, 6, 18	32, 23	19, 26	14, 10, 30
	QMIFS	12, 6, 18	32, 23	19, 21	28, 24, 25
	ReliefF	12, 6, 18	28, 14	21, 13	24, 23, 15

Table 4.10

Regression problem domain: ranking of feature selection methods for the first half of the data sets

<i>Data set</i>	<i>Method</i>	<i>Selected features</i>				<i>Average</i>
		3	5	7	10	
Communities	MIFS	1.5	2	4	5	3.5
	MRMR	1.5	2	1	2	1.6
	JMI	3	2	2.5	1	2.1
	QMIFS	4	4.5	2.5	3	3.4
	ReliefF	5	4.5	5	4	4.6
Parkinson telemonitoring	MIFS	4.5	4.5	5	1.5	3.9
	MRMR	4.5	4.5	1	1.5	2.9
	JMI	1.5	2.5	3.5	4.5	3
	QMIFS	3	2.5	3.5	4.5	3.4
	ReliefF	1.5	1	2	3	1.9
Wine quality	MIFS	3	5	3	3	3.5
	MRMR	1.5	1	3	3	2.1
	JMI	4	3	3	3	3.5
	QMIFS	1.5	3	3	3	2.6
	ReliefF	4	3	3	3	3.3
Housing	MIFS	5	4.5	4.5	3	4.3
	MRMR	2.5	4.5	4.5	5	4.1
	JMI	2.5	3	3	4	3.1
	QMIFS	4	2	2	1.5	2.4
	ReliefF	1	1	1	1.5	1.1

Table 4.11

Regression problem domain: ranking of feature selection methods for the second half of the datasets. The row with the overall averages includes also the data from Table 4.10.

Data set	Method	Selected features				Average
		3	5	7	10	
Web advertisement	MIFS	3.5	4	4	2	3.4
	MRMR	3.5	4	5	3	3.9
	JMI	2	2	3	5	3.0
	QMIFS	1	1	1	1	1.0
	ReliefF	5	4	2	4	3.8
Facebook	MIFS	4	4.5	4	1.5	3.5
	MRMR	1.5	1	1	1.5	1.3
	JMI	4	4.5	5	5	4.6
	QMIFS	1.5	2.5	2.5	3.5	2.5
	ReliefF	4	2.5	2.5	3.5	3.1
Blog	MIFS	3.5	2	3	3	2.8
	MRMR	5	3	3	1.5	3.1
	JMI	2	4.5	4	4.5	3.8
	QMIFS	1	1	1	1.5	1.1
	ReliefF	3.5	4.5	3	4.5	3.9
Bank	MIFS	3	3	3	3	3
	MRMR	3	3	3	3	3
	JMI	3	3	3	3	3
	QMIFS	3	3	3	3	3
	ReliefF	3	3	3	3	3
Average	MIFS		3.4			
	MRMR		2.8			
	JMI		3.3			
	QMIFS		2.4			
	ReliefF		3.1			

Table 4.12

Regression problem domain: values of RMSE measure for 7 selected features for the first half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Communities	MIFS	0.186
	MRMR	0.180
	JMI	0.183
	QMIFS	0.183
	ReliefF	0.191
	All features	0.190
	SE	0.0003
Parkinson telemonitoring	MIFS	8.93
	MRMR	8.23
	JMI	8.43
	QMIFS	8.42
	ReliefF	8.32
	All features	8.30
	SE	0.02
Wine quality	MIFS	0.772
	MRMR	0.773
	JMI	0.770
	QMIFS	0.771
	ReliefF	0.778
	All features	0.771
	SE	0.001
Housing	MIFS	5.12
	MRMR	5.06
	JMI	4.85
	QMIFS	4.61
	ReliefF	4.49
	All features	4.63
	SE	0.03

Table 4.13

Regression problem domain: values of RMSE measure for 7 selected features for the second half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Web advertisement	MIFS	3.668
	MRMR	3.703
	JMI	3.567
	QMIFS	3.174
	ReliefF	3.589
	All features	3.590
	SE	0.006
Facebook	MIFS	26.267
	MRMR	24.655
	JMI	25.750
	QMIFS	32.230
	ReliefF	25.447
	All features	26.912
	SE	0.136
Blog	MIFS	38.181
	MRMR	38.886
	JMI	41.266
	QMIFS	34.176
	ReliefF	39.577
	All features	37.896
	SE	0.287
Bank	MIFS	0.111
	MRMR	0.111
	JMI	0.111
	QMIFS	0.112
	ReliefF	0.116
	All features	0.117
	SE	0.001

Communities data set: Table 4.12 and the ones in the Appendix A show that the methods improve RMSE if we use more than three features to train the model. This is expected, since the number of features in the data set is quite large (100) and difficult

for the learning machine to tackle. Our method ranks second to last when selecting 3 or 5 features, but improves afterwards with RMSE comparable to the other three information-theoretic methods (second and third best at 7 and 10 selected features). The selected features across different methods are much more versatile on this data set, owing to the fact that there are a lot of input features to begin with.

Parkinson telemonitoring data set: There is only a small gain in the performance by using at least 7 features chosen by MRMR. The top 3 ranking features across all the methods are very similar, with only JMI and ReliefF offering 6 – 8% lower RMSE in comparison to others. However, our method performs equally well as JMI for five and more features.

Wine quality data set: In some cases feature selection offers an improvement in the regression performance even though the total number of features in the data set is only 11. Overall our method and MRMR are superior to MIFS, JMI, and ReliefF, selecting similar features and offering improvement over the baseline.

Housing data set: The baseline performs better here for the most part, but there are only 13 features in the data set, so the learning machine does not have a difficult task at training the model. Only our method and ReliefF show a small performance benefit compared to baseline when using at least 7 features. QMIFS achieves the second best overall performance among the five methods with an average rank of 2.4, lagging only behind ReliefF.

Web advertisement data set: Our method improves the model's performance dramatically compared to the baseline and other feature selection methods, which all exhibit similar behaviour. The number of input features in the data set is large enough to pose a difficult task to the learning machine, so it benefits considerably from feature selection, at least when QMIFS is used.

Facebook data set: Feature selection improves performance, except in the case of JMI, which surprisingly performs the worst on this data set. The best method in this case is MRMR. Our method performs quite well being second overall. MIFS and ReliefF lag behind somewhat except in the case of 10 selected features, where MIFS performs as good as MRMR and ReliefF is on par with QMIFS.

Blog data set: Here certain methods improve the performance over the baseline with others degrading it. QMIFS performs considerably better than the other methods in all cases with only MRMR achieving similar performance in the case of 10 selected features. Interestingly, JMI and ReliefF are the worst in this case.

Bank data set: Here the differences between the methods and the baseline are negligible. It seems that the simulation process behind these artificial data is quite simple, thus the Regression three can work with the whole feature set, without the need for feature selection as the number of features is rather small.

In terms of average RMSE ranks, our method outperforms the other three, achieving value of 2.4 across all test cases. MRMR takes the second place with an average rank of 2.8. These results suggest that without the possibility of using MDL to discretize the data, the other methods lag behind our approach. There are probably not many higher-order relations in the data, since JMI is comparable to MIFS in terms of overall performance. Obviously, the way underlying probability densities are estimated has a higher impact on the performance than the order of the method. We believe that QMIFS better distinguishes relations in the data than ad-hoc binning used in the other three methods. Interestingly ReliefF does not perform so well even though it avoids discretization, again this could be attributed to not quite optimal parameter selection for each data set.

Again we check if there are statistically significant differences between pairs all pairs of methods using the Friedman test [82, 83]. Similarly to the classification problem domain the test does not show significant differences between the methods. As mentioned before, this can be attributed to the fact that we need more data sets in order to establish significant differences between the methods. Nevertheless, we can definitely see that on individual data sets there are differences in the performance of the methods. We can argue that an optimal feature selection method for a certain data set should be selected through experimentation. Another aspect of selecting a method is its simplicity of usage, where QMIFS ranks favourably with its parameterless design and lack of the need for additional preprocessing steps like discretization.



Optimizations of QMIFS

5

Computational complexity of feature selection methods can be problematic when dealing with large data sets, which are common nowadays. To cope with these problem two approaches are possible, one is to reduce the computational complexity of the algorithm by means of some approximation or clever usage of data structures, the other is to employ massively parallel architectures like graphics processing units and coprocessors. In our case we try both options. We investigate the usage of Incomplete Cholesky decomposition to reduce the computational complexity of QMIFS and we implement QMIFS on two massively parallel architectures.

5.1 Incomplete Cholesky decomposition

Authors in [84] present an efficient approach to speed up the computation of I_{CS} with an insignificant loss of precision. The basic algorithm for computing I_{CS} has a time complexity of $O(n^2)$, where n is the number of samples. They use a greedy incomplete Cholesky decomposition algorithm in order to achieve the computational complexity of $O(nm^2)$, where m depends on the data and the desired precision ϵ . This approach is useful only when $m^2 < n$. In their work they achieve substantial time savings when dealing with common data sets, so we adopted their approach in the computation of I_{CS} .

The idea is to represent the estimation of quadratic Renyi entropy in terms of symmetric positive Gram matrix

$$\begin{aligned}\hat{H}_{R_2}^d(X) &= -\log \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2}\sigma}(x_i - x_j) \\ &= -\log \frac{1}{n^2} \mathbf{1}_n^T \mathbf{K}_{XX} \mathbf{1}_n ,\end{aligned}\tag{5.1}$$

where $\mathbf{1}_n$ is a vector of ones of size n and \mathbf{K}_{XX} is a matrix of the form

$$\mathbf{K}_{XX} = \begin{bmatrix} G_{\sqrt{2}\sigma}(x_1 - x_1) & \cdots & G_{\sqrt{2}\sigma}(x_1 - x_n) \\ \vdots & \ddots & \vdots \\ G_{\sqrt{2}\sigma}(x_n - x_1) & \cdots & G_{\sqrt{2}\sigma}(x_n - x_n) \end{bmatrix} .\tag{5.2}$$

We can express any symmetric positive definite matrix \mathbf{K} of size $n \times n$ in the form of Cholesky decomposition

$$\mathbf{K} = \mathbf{G}^T \mathbf{G} ,\tag{5.3}$$

where \mathbf{G} is a $n \times n$ lower triangular matrix. If the eigenvalues of \mathbf{K} drop rapidly, then the matrix can be approximated using incomplete Cholesky decomposition (ICD) by a $n \times m$, $m \leq n$ lower triangular matrix $\hat{\mathbf{G}}$.

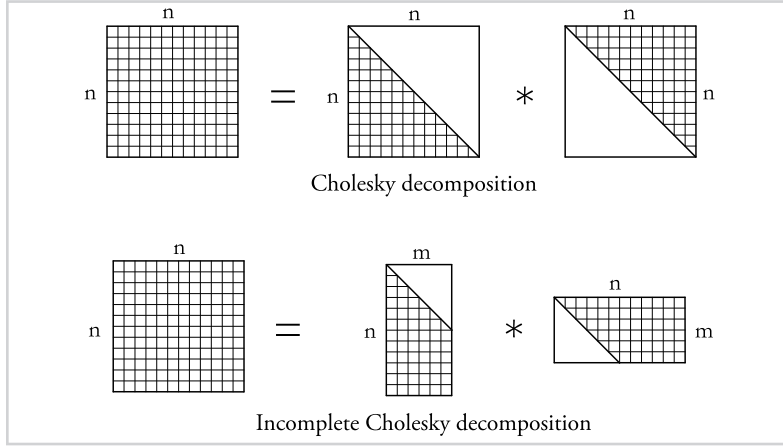


Figure 5.1

Complete and incomplete Cholesky decomposition of symmetric positive definite matrix

Quadratic Renyi entropy can then be estimated as

$$\hat{H}_{R_2}^d(X) = -\log \frac{1}{n^2} \mathbf{1}_n^T \hat{\mathbf{G}}_{XX}^T \hat{\mathbf{G}}_{XX} \mathbf{1}_n = -\log \frac{1}{n^2} \|\mathbf{1}_n^T \hat{\mathbf{G}}_{XX}\|_2^2. \quad (5.4)$$

Even though the time complexity of computing $\hat{H}_{R_2}^d(X)$ using $\hat{\mathbf{G}}$ is reduced from $O(n^2)$ to $O(nm^2)$, one has to find $\hat{\mathbf{G}}$. An efficient algorithm exists [85], which performs this task (Algorithm 2). It is basically a greedy optimization algorithm, that tries to minimize the trace of the residual $\mathbf{K} - \hat{\mathbf{G}}^T \hat{\mathbf{G}}$. The time complexity of the algorithm is $O(nm^2)$, thus the overall time complexity of estimating $\hat{H}_{R_2}^d(X)$ remains $O(nm^2)$.

We can use incomplete Cholesky decomposition to speed-up the computation of the QMIFS criterion function (4.1). As noted earlier the criterion is comprised of two terms; the relevancy term $I_{CS}(X_c; X_s; Z)$ and the redundancy term $I_{CS}(X_c; X_s)$.

First let's derive the equation for the computation of the redundancy term using ICD as presented in [84], since it involves only two variables. We will treat candidate feature X_c as variable X and the already selected feature X_s as variable Y in order to

Algorithm 2: Incomplete Cholesky decomposition

Data: Vector of feature's instances \mathbf{X} and desired precision ϵ *Result:* Incomplete lower triangular matrix $\hat{\mathbf{G}}$ *for* $k = 1$ *to* n *do*

$$\hat{\mathbf{G}}[k, 1] \leftarrow G_{\sqrt{2\sigma}}(\mathbf{X}[k] - \mathbf{X}[1])$$

$$\mathbf{D}[k] \leftarrow G_{\sqrt{2\sigma}}(0)$$

$$\mathbf{P}[k] \leftarrow k$$

end

$$i \leftarrow 1$$

$$s \leftarrow 1$$

while $i < n$ *do**if* $(\sum_{k=i}^n \mathbf{D}[k] < \epsilon)$ *then*
 *break**end*

$$j^* \leftarrow \operatorname{argmax}_{i \leq j \leq n} \mathbf{D}[j]$$

$$\mathbf{P}[i] \leftrightarrow \mathbf{P}[j^*] \quad // \text{exchange values}$$

for $k = 1$ *to* $i - 1$ *do*

$$\hat{\mathbf{G}}[i, k] \leftrightarrow \hat{\mathbf{G}}[j^*, k]$$

end

$$\mathbf{G}[i, i] \leftarrow \sqrt{\mathbf{D}[j^*, j^*]}$$

for $k = 1$ *to* $n - i$ *do*

$$\mathbf{K}[k] \leftarrow \sum_{l=1}^{i-1} \hat{\mathbf{G}}[i + k, l] \hat{\mathbf{G}}[i, l]$$

$$\mathbf{K}[k] \leftarrow (G_{\sqrt{2\sigma}}(\mathbf{X}[\mathbf{P}[i + k]] - \mathbf{X}[\mathbf{P}[i]]) - \mathbf{K}[k]) / \hat{\mathbf{G}}[i, i]$$

*end**for* $k = 1$ *to* $n - i$ *do*

$$\hat{\mathbf{G}}[i + k, i] \leftarrow \mathbf{K}[k]$$

end

$$s \leftarrow i$$

$$i \leftarrow i + 1$$

for $k = i$ *to* n *do*

$$\mathbf{D}[k] \leftarrow G_{\sqrt{2\sigma}}(0) - \sum_{l=1}^{i-1} \hat{\mathbf{G}}[k + i, l]^2$$

*end**end*Sort rows of $\hat{\mathbf{G}}$ according to \mathbf{P}

simplify notation.

$$\begin{aligned} I_{CS}(X_c; X_s) &= I_{CS}(X; Y) = D_{CS}(p_{XY}; p_X p_Y) \\ &= -\log \frac{V_C^2}{V_A V_B}, \end{aligned} \quad (5.5)$$

where

$$V_A = \int \int p_{XY}^2(x, y) dx dy, \quad (5.6)$$

$$V_B = \int \int p_X^2(x) p_Y^2(y) dx dy, \quad (5.7)$$

$$V_C = \int \int p_{XY}(x, y) p_X(x) p_Y(y) dx dy. \quad (5.8)$$

We can estimate each of the three terms V_A , V_B , V_C as follows

$$\begin{aligned} V_A &\approx \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2}\sigma}(x_i - x_j) G_{\sqrt{2}\sigma}(y_i - y_j) \\ &= \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \mathbf{K}_{XX}(i, j) \mathbf{K}_{YY}(i, j) \\ &\approx \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \left(\sum_{k=1}^{m_x} \hat{\mathbf{G}}_{XX}(i, k) \hat{\mathbf{G}}_{XX}^T(k, j) \sum_{l=1}^{m_y} \hat{\mathbf{G}}_{YY}(i, l) \hat{\mathbf{G}}_{YY}^T(l, j) \right) \\ &= \frac{1}{n^2} \sum_{k=1}^{m_x} \sum_{l=1}^{m_y} \left(\sum_{i=1}^n \hat{\mathbf{G}}_{XX}^T(k, i) \hat{\mathbf{G}}_{YY}(i, l) \sum_{j=1}^n \hat{\mathbf{G}}_{YY}^T(l, j) \hat{\mathbf{G}}_{XX}(j, k) \right) \\ &= \frac{1}{n^2} \mathbf{1}_{d_x}^T \left(\hat{\mathbf{G}}_{XX}^T \hat{\mathbf{G}}_{YY} \odot \hat{\mathbf{G}}_{XX}^T \hat{\mathbf{G}}_{YY} \right) \mathbf{1}_{d_y} \\ &= \hat{V}_A, \end{aligned} \quad (5.9)$$

where \odot is the Hadamard operator (element-wise multiplication).

$$\begin{aligned} V_B &\approx \frac{1}{n^4} \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2}\sigma}(x_i - x_j) \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2}\sigma}(y_i - y_j) \\ &\approx \frac{1}{n^4} \|\mathbf{1}_n^T \hat{\mathbf{G}}_{XX}\|_2^2 \|\mathbf{1}_n^T \hat{\mathbf{G}}_{YY}\|_2^2 \\ &= \hat{V}_B, \end{aligned} \quad (5.10)$$

$$\begin{aligned}
V_C &\approx \frac{1}{n^3} \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n G_{\sqrt{2\sigma}}(x_i - x_j) G_{\sqrt{2\sigma}}(y_i - y_k) \\
&= \frac{1}{n^3} \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \mathbf{K}_{XX}(i, j) \mathbf{K}_{YY}(i, k) \\
&= \frac{1}{n^3} \mathbf{1}_n^T \mathbf{K}_{XX} \mathbf{K}_{YY} \mathbf{1}_n \\
&\approx \frac{1}{n^3} (\mathbf{1}_n^T \hat{\mathbf{G}}_{XX}) (\hat{\mathbf{G}}_{XX}^T \hat{\mathbf{G}}_{YY}) (\hat{\mathbf{G}}_{YY}^T \mathbf{1}_n) \\
&= \hat{V}_C.
\end{aligned} \tag{5.11}$$

If we now substitute the three terms V_A , V_B , and V_C in Eq. 5.5 with their estimates \hat{V}_A , \hat{V}_B , and \hat{V}_C , we obtain the estimator [84] for the redundancy part of the criterion function

$$\begin{aligned}
\hat{I}_{CS}(X_c; X_s) &= \hat{I}_{CS}(X; Y) = \\
&= -\log \frac{\hat{V}_C^2}{\hat{V}_A \hat{V}_B}.
\end{aligned} \tag{5.12}$$

From the above equations we see that the most time consuming operation is the multiplication of $\hat{\mathbf{G}}_{XX}^T \hat{\mathbf{G}}_{YY}$. This operation requires on the order of $O(nm_x m_y)$ operations, which makes the computational complexity of computing the redundancy term $O(nm^2)$, where $m = \max(m_x, m_y)$.

Now we can deal with the more complicated relevancy part of the criterion function. It involves three variables: the candidate feature X_c , the already selected feature X_s and the output Z . Again we simplify the notation by substituting X_c and X_s with X and Y .

$$\begin{aligned}
I_{CS}(X_c; X_s; Z) &= I_{CS}(X, Y; Z) = D_{CS}(p_{XYZ}; p_X p_Y p_Z) \\
&= -\log \frac{U_C^2}{U_A U_B},
\end{aligned} \tag{5.13}$$

where

$$U_A = \int \int \int p_{XYZ}^2(x, y, z) dx dy dz, \quad (5.14)$$

$$U_B = \int \int \int p_X^2(x) p_Y^2(y) p_Z^2(z) dx dy dz, \quad (5.15)$$

$$U_C = \int \int \int p_{XYZ}(x, y, z) p_X(x) p_Y(y) p_Z(z) dx dy dz. \quad (5.16)$$

The estimation of the three terms U_A , U_B , U_C goes as follows

$$\begin{aligned} U_A &\approx \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2\sigma}}(x_i - x_j) G_{\sqrt{2\sigma}}(y_i - y_j) G_{\sqrt{2\sigma}}(z_i - z_j) \\ &= \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \mathbf{K}_{XX}(i, j) \mathbf{K}_{YY}(i, j) \mathbf{K}_{ZZ}(i, j) \\ &\approx \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \left(\sum_{k=1}^{m_x} \hat{\mathbf{G}}_{XX}(i, k) \hat{\mathbf{G}}_{XX}^T(k, j) \sum_{l=1}^{m_y} \hat{\mathbf{G}}_{YY}(i, l) \hat{\mathbf{G}}_{YY}^T(l, j) \sum_{t=1}^{m_z} \hat{\mathbf{G}}_{ZZ}(i, t) \hat{\mathbf{G}}_{ZZ}^T(t, j) \right) \\ &= \frac{1}{n^2} \sum_{t=1}^{m_z} \sum_{k=1}^{m_x} \sum_{l=1}^{m_y} \left(\sum_{i=1}^n \hat{\mathbf{G}}_{XX}(i, k) \hat{\mathbf{G}}_{YY}(i, l) \hat{\mathbf{G}}_{ZZ}(i, t) \sum_{j=1}^n \hat{\mathbf{G}}_{XX}(j, k) \hat{\mathbf{G}}_{YY}(j, l) \hat{\mathbf{G}}_{ZZ}(j, t) \right) \\ &= \frac{1}{n^2} \sum_{t=1}^{m_z} \mathbf{1}_{m_y}^T \left((\hat{\mathbf{G}}_{YY}^T \odot \hat{\mathbf{G}}_{ZZ}^T(t) \mathbf{1}_{m_y}^T) \hat{\mathbf{G}}_{XX} \right) \odot \left(\hat{\mathbf{G}}_{YY}^T \odot \hat{\mathbf{G}}_{ZZ}^T(t) \mathbf{1}_{m_y}^T \right) \hat{\mathbf{G}}_{XX} \mathbf{1}_{m_x} \\ &= \hat{U}_A, \end{aligned} \quad (5.17)$$

where $\hat{\mathbf{G}}_{ZZ}(t)$ represents row t of matrix $\hat{\mathbf{G}}_{ZZ}$,

$$\begin{aligned} U_B &\approx \frac{1}{n^6} \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2\sigma}}(x_i - x_j) \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2\sigma}}(y_i - y_j) \sum_{i=1}^n \sum_{j=1}^n G_{\sqrt{2\sigma}}(z_i - z_j) \\ &\approx \frac{1}{n^6} \|\mathbf{1}_n^T \hat{\mathbf{G}}_{XX}\|_2^2 \|\mathbf{1}_n^T \hat{\mathbf{G}}_{YY}\|_2^2 \|\mathbf{1}_n^T \hat{\mathbf{G}}_{ZZ}\|_2^2 \\ &= \hat{U}_B. \end{aligned} \quad (5.18)$$

$$\begin{aligned}
U_C &\approx \frac{1}{n^4} \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{l=1}^n G_{\sqrt{2}\sigma}(x_i - x_j) G_{\sqrt{2}\sigma}(y_i - y_k) G_{\sqrt{2}\sigma}(z_i - z_l) \\
&= \frac{1}{n^4} \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{l=1}^n \mathbf{K}_{XX}(i, j) \mathbf{K}_{YY}(i, k) \mathbf{K}_{ZZ}(i, l) \\
&= \frac{1}{n^4} \mathbf{1}_n^T \mathbf{K}_{XX} \mathbf{K}_{YY} \mathbf{K}_{ZZ} \mathbf{1}_n \\
&\approx \frac{1}{n^4} \left((\mathbf{1}_n^T \hat{\mathbf{G}}_{XX}) \hat{\mathbf{G}}_{XX}^T \right) \odot \left((\mathbf{1}_n^T \hat{\mathbf{G}}_{YY}) \hat{\mathbf{G}}_{YY}^T \right) \odot \left((\mathbf{1}_n^T \hat{\mathbf{G}}_{ZZ}) \hat{\mathbf{G}}_{ZZ}^T \right) \mathbf{1}_n \\
&= \hat{U}_C,
\end{aligned} \tag{5.19}$$

Deriving \hat{U}_B for the three variable case is the most straightforward, since the double sums across the variables are independent of each other. Terms \hat{U}_A and \hat{U}_C are a little bit more complicated. In both cases we start with basic formula and transform it along similar lines as in the two-variable case, but the final step is much more convoluted. We must be careful to perform the matrix and vector operations in the right order to get the correct result and minimize the number of operations. Substitution of the three terms U_A , U_B , and U_C in Eq. 5.13 with their respective estimates, yields the estimator for the relevancy part of the criterion function

$$\begin{aligned}
\hat{I}_{CS}(X_c; X_s; Z) &= \hat{I}_{CS}(X; Y; Z) = \\
&= -\log \frac{\hat{U}_C^2}{\hat{U}_A \hat{U}_B}.
\end{aligned} \tag{5.20}$$

Here, the most time consuming operation is the computation of \hat{U}_A . It requires on the order of $O(nm_x m_y m_z)$ operations, which means that the computational complexity of the criterion function is roughly $O(nm^3)$, where $m = \max(m_x, m_y, m_z)$.

5.2 Exploiting custom hardware

Modern commodity computers are heterogeneous platforms with many different types of computational units, including central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), coprocessors, and custom acceleration logic (Fig. 5.2). Today's CPUs contain from two to twenty-four cores, each capable of executing multiple instructions per clock cycle. Assisting the CPU, graphics processing units usually render 3D graphics, but can also provide a general-purpose computing platform.

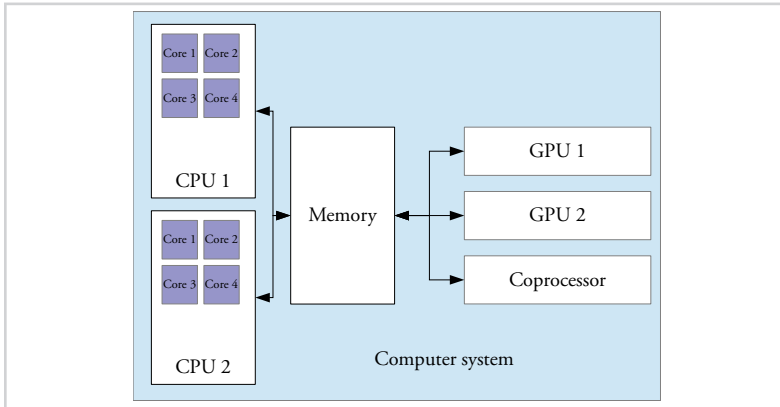


Figure 5.2

Simple scheme of a modern commodity computer with multiple multi-core CPUs, GPUs, and a coprocessor.

In many fields of engineering and research constant improvements in data acquisition techniques lead to the production of large amounts of observational data. However, analysis and modelling of very large data sets quickly poses an unbridgeable obstacle for algorithms that don't effectively exploit the underlying computer hardware.

5.2.1 Parallel computing on GPUs

GPUs are designed as massively parallel processing units offering substantial computing power. The accessibility of general-purpose GPUs in commodity laptop and desktop computers has generated a wide interest in many research fields, ranging from simulations of physical processes [86], modelling of complex systems [87], image and video processing [88] to artificial intelligence and data mining [89]. GPUs are the most powerful computational hardware available at an affordable price [90]. They are often the tool of choice when tackling computationally intensive problems while being without access to specialized hardware or high-performance computing systems.

GPU architecture

The technology behind designing and building modern CPUs has advanced greatly since the birth of general-purpose electronic computers, causing dramatic performance and efficiency improvements with each new generation of CPUs. However, the fundamental design is still based on the classic, more than half a century old, von-Neumann architecture [91]. As presented in Figure 5.3, a CPU consists of a control unit, which

interprets instructions and controls communication between the memory connected to the data bus and the arithmetic logic (ALU) unit. The latter is the work-horse of the CPU, tasked with executing instructions provided by the control unit. Modern CPUs also include a small amount of on-chip memory (cache) and contain multiple ALUs. Multi-core processors go a step further and group together multiple CPUs (cores) with additional control logic and caches, all on a single chip.

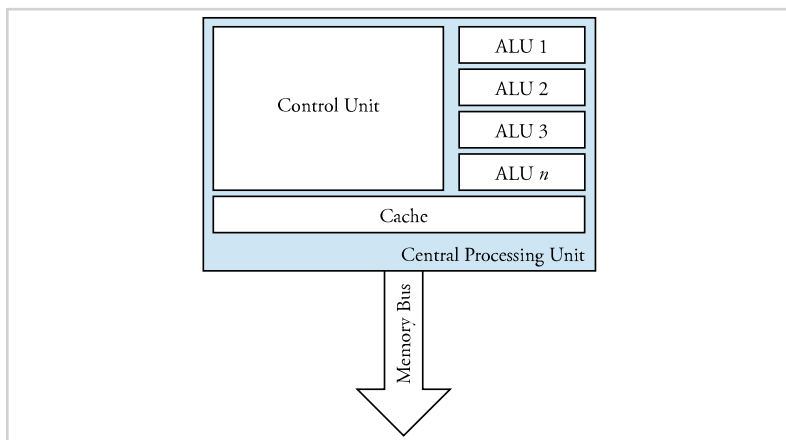


Figure 5.3

Modified Von-Neumann architecture.

GPUs follow an essentially different architecture paradigm. In the beginning they were designed as highly specialized hardware suitable for processing of the graphics pipeline. However, their design has moved to a more general architecture capable of other computational tasks [92]. The basic architecture of a modern GPU is presented in Fig. 5.4. It consists of several multiprocessors (SM) connected to a common on-chip memory (L2 cache). Each of the SMs is comprised of numerous simple processing cores which share a common control logic, cache, a chunk of programmably accessible shared memory, and a register file. All multiprocessors have access to the main graphics memory through a high bandwidth bus. For example, currently top of the line Nvidia GPU Titan Xp has 60 SMs with each of them having 64 processing cores, yielding the total of 3840 cores. Each of the SMs has 64 KB of shared memory and 24 KB of L1 cache. The L2 cache amounts to 4 MB and the main graphics memory up to 12 GB.

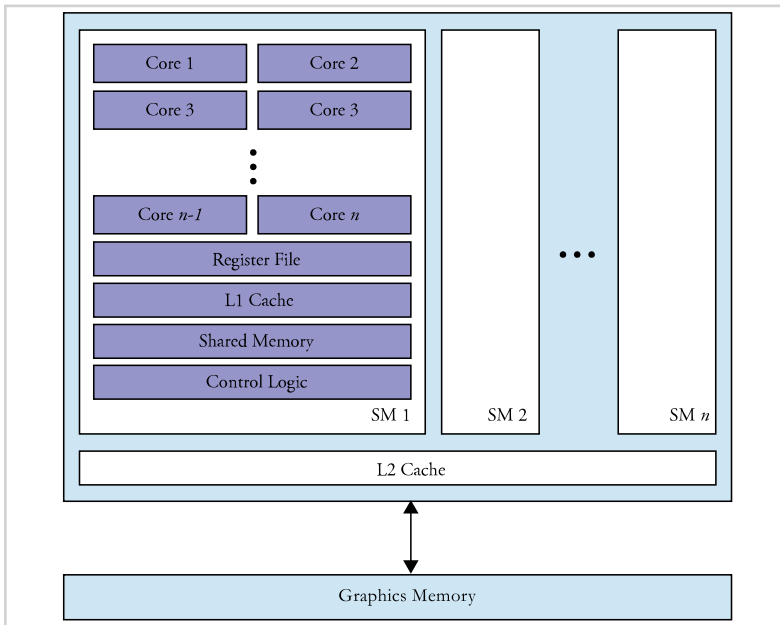


Figure 5.4

Architecture of a modern GPU.

The high number of processing units (cores) in modern GPUs leads to an impressive raw computational power. They are designed to carry out massive amounts of floating point calculations in parallel by executing a large number of threads with similar instructions thus minimizing control logic and long-latency memory accesses. This approach favours processing of large amounts of data in a highly parallel manner and performs badly when exposed to many different smaller tasks. This is in contrast to CPUs which can deal with smaller tasks effectively, but the memory bus bottleneck considerably slows down processing of large amounts of data [8].

Of course the architecture and the terminology describing GPU differs to some extent between different vendors, but the main ideas are the same. In our case we used the Nvidia terminology and architecture as an example.

GPU programming model

A programmer sees the GPU as a parallel coprocessor and can use it to speed-up computationally intensive parts of the algorithm. He has to identify the parallelism in the

algorithm and divide the processing into many small, independent subtasks, that can be run in parallel.

Nvidia defined its own programming model, called CUDA [9], which is tightly bound to the underlying GPU architecture. The programmer has to be aware of strengths and limitations of GPUs in order to fully exploit their raw computational power.

A program which uses GPUs (Fig. 5.6) starts on the CPU (host) and is executed serially until a section of the program meant for the GPU (device) is encountered. If some data needs to be processed on the GPU, then it has to be copied to the graphics memory of the GPU – in the programming model referred to as global memory. The GPU section of the program (kernel) then starts to execute on the GPU. After the kernel execution has finished, the results (if any) are transferred back to the hosts main memory. There are some additional types of memory defined in the programming model, namely texture memory, local memory, and constant memory. Physically they all reside in the main graphics memory, but are only useful in special cases and we won't be dealing with them in the scope of this work. For more details see [93].

The kernel is executed in parallel by numerous independent threads, which are organized into blocks. These blocks are arranged into a grid, that facilitates distribution of work across blocks. All the threads in a grid execute the same kernel. However the programmer has mechanisms (thread and block identifiers) to distinguish between different blocks and threads and can branch the execution depending on the current block and thread id. Threads and blocks can be arranged into a one-, two-, or three-dimensional structure depending on the problem domain. Fig. 5.5 shows a simple kernel written in C using CUDA.

```
//serial code
void add(float *a, float *b, float *c, int n)
{
    for(int i = 0; i < n; i++)
        c[i] = a[i] + b[i];
}

//CUDA kernel
__global__ void add(float *a, float *b, float *c, int n)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    while(i < n)
    {
        c[i] = a[i] + b[i];
        i += gridDim.x * blockDim.x;
    }
}
```

Figure 5.5

Simple code snippet written in C to show the differences between ordinary serial code and code that is meant for execution on the GPU (CUDA kernel). The code listed performs an element-wise addition of two arrays and stores the result in the third array.

Each thread has its own set of registers and can access the shared memory and the global memory. The shared memory resides in the multiprocessor and is common to all threads of a block. This memory acts as a fast cache, but it is different from the normal CPU cache because the programmer can explicitly define which data should be kept in it.

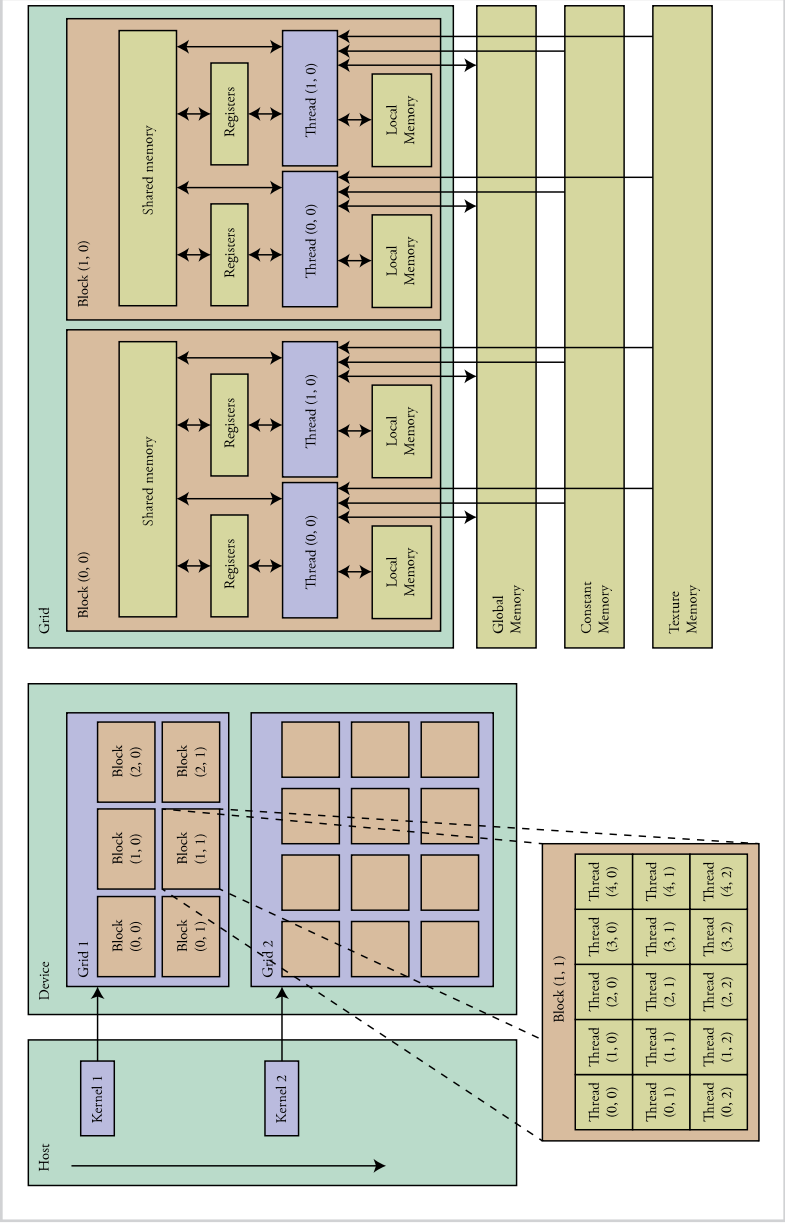


Figure 5.6
Nvidia CUDA program-
ming model.

To write efficient code for the GPU, the programmer must follow certain rules [94]. First, to utilize all of the processing resources of a GPU, there should be enough threads running at any given time. As different Nvidia GPUs have different number of multiprocessors, cores, and other resources, and as the requirements per thread depend on the algorithm, this number can vary considerably. Second, the number of divergent branches inside a thread block should be kept at minimum, otherwise the code is forced to execute serially for each branch taken by different threads. Third, the code should maximize register and shared memory usage, since the access latency to this two types of memories is very small in comparison to the global memory. Next, the data transfer between graphics memory and the computer main memory should be kept as low as possible to avoid bottlenecks. Finally, the memory access pattern must be taken into account. The time required to access the global memory can be reduced, if threads request a single continuous segment of the global memory. Neglecting these rules can seriously cripple the algorithms performance.

Different tools are available for programming GPUs. Nvidia offers the CUDA toolkit [95] for programming its own products. It includes a proprietary compiler and a set of libraries that extend the C++ syntax with parallel programming constructs. Another popular option is the OpenCL framework [96]. It supports hardware from different vendors but usually lags slightly in terms of performance when compared to specialized development kits such as CUDA.

QMIFS on graphics processing unit

Usage of GPUs in the area of machine learning has become widespread in the recent years. They are especially useful to speed up training of neural networks [97], and are the tool of choice in deep learning applications [98]. Researchers have created numerous open source libraries, which simplify usage of GPUs in such applications [97, 99, 100]. Problems where big chunks of data are being analyzed often utilize GPUs to bring down the execution times to workable levels. Examples are analysis of genetic data [101, 102] and social network data [103].

GPUs have been also used to speed up feature selection methods [104–106]. In Chapter 4 we adopted the incomplete Cholesky decomposition to reduce the execution time of QMIFS. Another possible approach would be to parallelize the algorithm and modify it for execution on GPUs. In one of our previous works [107] we explored the possibility of using heterogeneous computer systems in order to speed up

searching for interactions between single nucleotide polymorphisms (SNPs) in genetic data, with the goal of detecting possible genetic causes for certain illnesses. We used mutual information (2.9) and information interaction (2.17) to discover second order interactions between features (in this case SNPs) with regards to the output (subjects diagnosis). Due to the extent of the data a parallel approach to the analysis was crucial. The computation of mutual information and information interaction between different features and the output is data-independent. Consequently, we can design effective parallel algorithms for their computation.

The analysis involved computation of $I(X_i; Z)$, $I(X_j; Z)$, and $I(X_i; X_j; Z)$ for every pair of features (X_i, X_j) , $i \neq j$ and the output Z . Since mutual information and interaction information is symmetric, we need to perform $N(N-1)/2$ evaluations of these measures per data set, where N is the total number of features. The obvious approach to parallelizing the analysis, is to partition the evaluations of these measures into chunks and distribute them between different computational units in a computer system. The algorithm we designed is capable of distributing work across multi-core CPUs, GPUs, and XeonPhi coprocessors. The measurements of execution times showed that GPUs are the fastest for performing these kind of analyses.

If we now consider our feature selection method – QMIFS, we note that it suggests a similar approach to feature selection. The criterion function (4.1) involves computation of quadratic mutual information for all possible pairs of candidate features X_c and already selected features X_s . The evaluations are independent of each other and as such suitable for parallelization. The most promising is the usage of GPUs, due to the similarities between QMIFS and the method mentioned in the previous paragraph.

In order to adopt the algorithm for execution on a GPU, we need to partition the problem so that it can be efficiently mapped to the GPU threads. As mentioned before, GPUs execute very large number of simple threads in parallel. The most logical approach is to assign one evaluation of the quadratic mutual information to one thread in order to maximize the workload. To do this we modified Algorithm 1 so that it computes quadratic mutual information between all possible pairs of features and the output before performing actual selection of features. This part of the algorithm is the most computationally intensive and is thus executed in parallel on the GPU. Everything else is executed serially in the same way as before. This approach means that some values of quadratic mutual information, which were computed on the GPU may not be used in the process of feature selection. The advantages are in the reduced GPU

management and number of memory transfers, which would be present if calculations on the GPU would be performed on demand during the process of feature selection. Algorithm 3 shows the pseudo-code of the parallel QMIFS algorithm.

An important consideration is the minimization of data transfer between the graphics memory and main system memory. At the beginning we have to transfer the whole data set of features and the output to the GPU. After the computations on the GPU have completed, we obtain the matrix \mathbf{U} of the feature scoring heuristic, where $U_{ij} = I_{CS}(X_i; X_j; Z) - I_{CS}(X_i; X_j)$. The dimensions of the matrix are $N \times N$, but due to the fact that quadratic mutual information is symmetric and that the diagonal elements of the matrix \mathbf{U} are not computed (see Algorithm 3), we are basically dealing with an upper-triangular matrix with $N(N - 1)/2$ relevant values. Therefore it would be wasteful to keep the whole matrix in the memory and transfer it from the graphics memory to the system memory. To avoid this we created an index transformation formula (5.21), which enables us to store and access values of the upper-triangular matrix \mathbf{U} in a compact one-dimensional array. The formula works by translating between the one-dimensional index $k = 0, \dots, N(N - 1)/2 - 1$ and two-dimensional indices (c, s) of the matrix \mathbf{U} , which also identify the candidate feature X_c and the already selected feature X_s .

$$\begin{aligned} c &= N - (t + 2) \\ s &= i - N(N - 1)/2 + (t + 1)(t + 2)/2 + c + 1, \end{aligned} \quad (5.21)$$

where

$$t = \frac{\sqrt{4N(N - 1) - 8(k + 1) + 1} - 1}{2}. \quad (5.22)$$

Algorithm 3: Parallel quadratic mutual information feature selection

Data: Set of candidate features \mathbf{X}_C and output Z

Result: Set of selected features indices \mathbf{S}

Standardize \mathbf{X}_C and Z

$\mathbf{X}_S \leftarrow \emptyset$

$\mathbf{S} \leftarrow \emptyset$

$M \leftarrow 0$

// The following for loop is suitable for execution on the GPU

for $X_i, X_j \in \mathbf{X}_C; j > i$ in parallel

$U_{ij} \leftarrow I_{CS}(X_i, X_j; Z) - I_{CS}(X_i; X_j)$

end

while stopping condition not met do

$S_{\max} = 0$

for $X_c \in \mathbf{X}_C$ do

if $M > 0$ then

$S_c \leftarrow \sum_{s=1}^M U_{sc}$

else

$S_c \leftarrow I_{CS}(X_c; Z)$

end

if $S_c > S_{\max}$ then

$S_{\max} \leftarrow S_c$

$X_{\max} \leftarrow X_c$

$c_{\max} \leftarrow c$

end

end

$\mathbf{X}_C \leftarrow \mathbf{X}_C / X_{\max}$

$\mathbf{X}_S \leftarrow \mathbf{X}_S \cup X_{\max}$

$\mathbf{S} \leftarrow \mathbf{S} \cup c_{\max}$

$M \leftarrow M + 1$

end

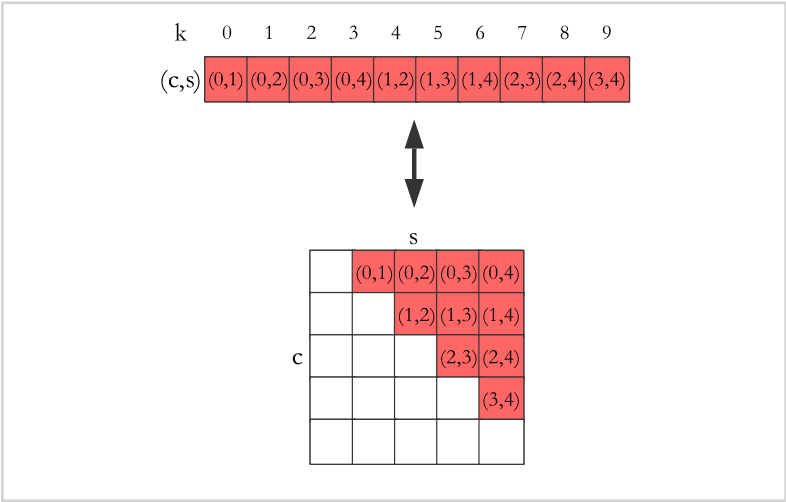


Figure 5.7
Transformation between linear indices k and pairs of candidate and already selected features (c,s) in the five feature case.

An example of how the transformation works in the case of 5 features is shown in Fig. 5.7.

5.2.2 Parallel computing on the MIC coprocessors

In 2010, Intel announced the Intel Many integrated core architecture (MIC) – a family of general-purpose, many-core coprocessors to compete with GPUs at speeding up scientific applications [108]. The architecture is meant to compete with GPUs specialized in general-purpose computing. The design follows a different approach in comparison to GPUs. Coprocessors consists of many simple, but fully functional processor cores derived from the Intel Pentium architecture. The idea is to improve programmability of such coprocessing devices by supporting a well-established shared-memory execution model based on the classical Intel architecture. Intel improved the original design by adding a 512-bit wide vector unit and Hyper-Threading Technology. This enables MIC coprocessors to achieve similar theoretical performance as modern GPU [109]. Some of the top supercomputers such as China's Tianhe-2 use clusters of these coprocessors to achieve petaflop theoretical performance while keeping the power consumption within manageable limits [110].

MIC architecture

The basic MIC architecture is presented in Fig. 5.8. The current implementation of the MIC architecture is in the form of the Xeon Phi series of coprocessors. The model 5510P, which we used in this study, includes sixty cores interconnected with a bi-directional ring bus. Each core is capable of running four threads in parallel. The design uses typical cache structure of per-core level-one (L1) and level-two (L2) caches. The shared L2 cache uses the high-bandwidth ring bus for fast on-chip communication. The cores fetch data from the 8 GB of on-board RAM and communicate with the host CPU through the PCIe bus. In comparison to GPUs, each core on a Xeon Phi can efficiently execute the code even if threads do not follow the same program path. This makes it suitable for a wider range of problems, including multiplications of sparse matrices [111], and operations on trees and graphs [112].

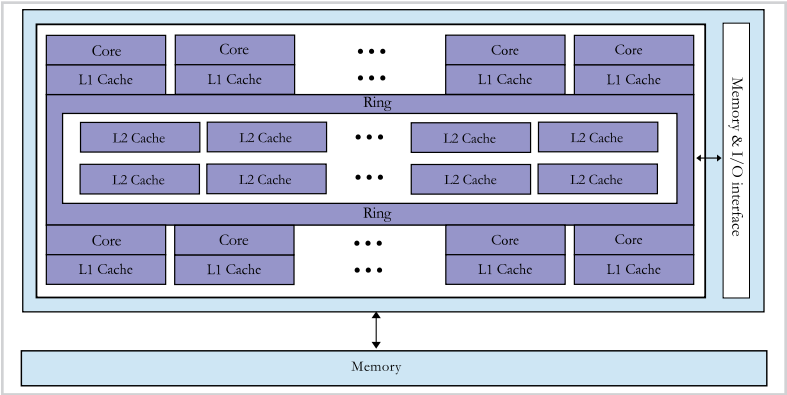


Figure 5.8
Architecture of a MIC
coprocessor.

MIC programming model

Based on Intel architecture, MIC supports already established programming models available for traditional processors. Intel provides a compiler suite and all the tools needed to exploit the hardware [113]. It has built-in support for Fortran and C/C++. The code can be parallelized using OpenMP directives, MPI library, or some other approach and compiled for the MIC architecture. Resulting applications can then run only on the Xeon Phi coprocessors. Xeon Phi is essentially a mini computer within the host. A specialized version of Linux operating system runs on the device, with

the user being able to log on and perform tasks directly on the coprocessor. Another, more general way to specify parallel execution is to use offload constructs along with OpenMP to mark the data and the code to be transferred and executed on the Xeon Phi. All other parts of the program will run normally on the host computer CPU. A third possibility is to use OpenCL framework in the same manner as with GPUs. MIC development tools facilitate data management through compiler directives. The example in Fig. 5.9 demonstrates this programming paradigm. It performs the same operation as the snippet from Fig. 5.5. The programmer marks the data and the code that is needed on the coprocessor. All memory allocations and transfers are done implicitly. The MIC code looks much cleaner and easier to handle than CUDA code. To obtain best performance, the programmer must tailor the algorithms to fully utilize the vector unit. The Intel compiler automatically vectorizes the code where possible, to make best use of the vector unit. of course the programmer can help in this regard by using compiler directives or he can even perform the whole vectorization manually through the use of intrinsic functions.

```
//MIC offload
void add(float *a, float *b, float *c, int n)
{
    #pragma offload target(mic) in(a,b) out(c)
    #pragma omp parallel for
    for (int i=0; i<n; i++)
        c[i] = a[i] + b[i];
}
```

Figure 5.9

Simple code snippet written in C to show how compiler directives and OpenMP can be employed to execute parts of the code on Xeon Phi. The code listed performs an element-wise addition of two arrays and stores the result in the third array.

QMIFS on MIC

Although not so common and affordable as GPUs, MIC coprocessors have also found their way into the scientific community and are especially useful in supercomputing centres, due to their seamless integration and less demanding programming interface. Although their main usage lies in large scale simulation of physical systems, they have been used in machine learning tasks especially in the field of deep neural networks [114, 115].

Given the nature of QMIFS and our previous experience regarding Xeon Phi coprocessors [107], where they proved to be faster than multi-core processors, it would be meaningful to try implement QMIFS on such architecture. We can adopt the

same approach in implementing QMIFS on the Xeon Phi as in the case of the GPU. Essentially the same general approach and algorithm 3 can be used with all the considerations, such as minimizing data transfers also valid here. We adopt the offload paradigm for our implementation of QMIFS, where only parts of code are executed on the coprocessor. The parallelization is done using OpenMP directives with only the data set being transferred to the coprocessor and the resulting upper-triangular matrix \mathbf{U} (in compact form) transferred back to the host memory.

5.2.3 QMIFS on a heterogeneous architecture

Looking at the parallel QMIFS algorithm (3) and various implementations it seems reasonable to combine all the implementations in order to exploit all of the system hardware depicted in Fig. 5.2. The idea is simple, at the beginning distribute the work of computing the matrix \mathbf{U} between all implementations and then gather the results. The architecture of such approach is depicted in Fig. 5.10. Each of the participating implementations receives the data set and is assigned a continuous chunk of matrix \mathbf{U} for which it has to compute the values.

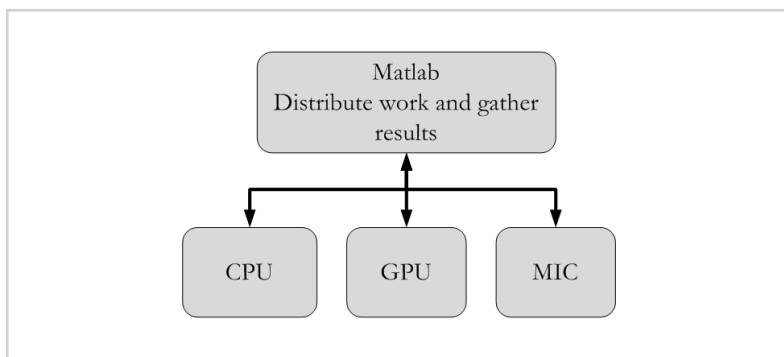


Figure 5.10

Scheme of the heterogeneous implementation of QMIFS. The software in Matlab assigns work to each implementation and at the end gathers the results.

The main issue is to distribute work according to processing power of each implementation. This needs to be done in order to keep all the hardware busy during the analysis and consequently obtain the lowest execution time. This can be done in multiple ways. One can perform a short benchmark at the beginning to establish the performance of the hardware. This adds additional overhead to the computation, but if the data set is large it is not a big issue. Of course if the hardware and its performance

is known in advance the benchmark can be avoided and the sizes of the workloads can be set in advance. Another option is to assign small chunks to each implementation at the beginning and then, according to the measured performance, adjust the sizes of chunks that follow. If needed the adjustment can be performed multiple times during execution. This looks like the most promising approach although it has its drawbacks. The main being the fact that GPU kernels and MIC offloads need to be called multiple times, and that the utilization of these two architectures is low at the beginning. In our case, as proof of concept, we implement only the option where we set up the workloads in advance based on the computational capability of the implementations.

5.3 *Results and discussion*

To evaluate the effectiveness of the QMIFS optimizations we first perform experiments on the data sets from Chapter 4 and establish how QMIFS and its optimizations fare with regards to execution time against the other methods (MIFS, MRMR, JMI, and ReliefF), which we evaluated in the previous chapter. In each data set we select half of the features. We test the methods on a workstation with two six-core Intel Xeon E5-2620 2.00 GHz CPUs capable of running up to twenty-four threads in parallel, 64 GB of RAM, Nvidia Tesla K20 general-purpose computing card with 5 GB of RAM and an Intel Xeon Phi 5110P coprocessor with 8 GB of RAM. The operating system is CentOS 6.4. The K20 GPU features 13 multiprocessors with 192 cores per SM for a total of 2496 cores. In the experiments we include four implementations of QMIFS:

1. QMIFS-naive, which uses direct computation of quadratic mutual information,
2. QMIFS-ICD, which uses the incomplete Cholesky decomposition, and
3. QMIFS-GPU, which is the GPU implementation of QMIFS-naive.
4. QMIFS-MIC, which is the MIC implementation of QMIFS-naive.

The base of all four algorithms is implemented in Matlab R2016a. The computationally intensive parts of QMIFS-naive are written in C programming language. The GPU programming is done in CUDA C and the MIC implementation is done in C using OpenMP directives. There is no need to port computationally intensive parts of QMIFS-ICD to C since it mostly uses matrix operations, which are very fast in Matlab. Each measurement was performed ten times with the execution times averaged to

obtain the final result, except for QMIFS-naive in certain cases where the execution times were very long (ie. Facebook and Blog data sets), where we did only one run. The relative error of the measurements is around 2% in all cases.

Table 5.1

Classification problem domain: execution times for selecting half of the features present in the data set for various methods and QMIFS optimizations – first half of the data sets.

<i>Data set</i>	<i>Method</i>	<i>Execution times [s]</i>
Chess Instances: 1000 Features: 36	MIFS	2.5
	MRMR	2.4
	JMI	2.8
	ReliefF	0.8
	QMIFS-naive	44.7
	QMIFS-ICD	0.3
	QMIFS-GPU	0.3
	QMIFS-MIC	0.5
Breast cancer Instances: 569 Features: 30	MIFS	1.3
	MRMR	1.2
	JMI	1.5
	ReliefF	0.6
	QMIFS-naive	13.6
	QMIFS-ICD	0.5
	QMIFS-GPU	1.2
	QMIFS-MIC	1.5
Ionosphere Instances: 351 Features: 34	MIFS	1.1
	MRMR	1.2
	JMI	1.1
	ReliefF	0.4
	QMIFS-naive	6.5
	QMIFS-ICD	0.4
	QMIFS-GPU	0.7
	QMIFS-MIC	0.9
Sonar Instances: 208 Features: 60	MIFS	1.2
	MRMR	1.2
	JMI	1.6
	ReliefF	0.6
	QMIFS-naive	7.8
	QMIFS-ICD	0.5
	QMIFS-GPU	0.4
	QMIFS-MIC	0.5

Table 5.2

Classification problem domain: execution times for selecting half of the features present in the data set for various methods and QMIFS optimizations – second half of the data sets.

<i>Data set</i>	<i>Method</i>	<i>Execution times [s]</i>
Wine Instances: 178 Features: 13	MIFS	0.4
	MRMR	0.4
	JMI	0.5
	ReliefF	0.4
	QMIFS-naive	0.6
	QMIFS-ICD	0.3
	QMIFS-GPU	0.3
	QMIFS-MIC	0.3
Cervical cancer Instances: 58 Features: 714	MIFS	22.1
	MRMR	22.2
	JMI	34.3
	ReliefF	0.5
	QMIFS-naive	86.2
	QMIFS-ICD	9.5
	QMIFS-GPU	1.9
	QMIFS-MIC	2.1
Spine Instances: 310 Features: 12	MIFS	0.6
	MRMR	0.5
	JMI	0.6
	ReliefF	0.4
	QMIFS-naive	0.8
	QMIFS-ICD	0.3
	QMIFS-GPU	0.4
	QMIFS-MIC	0.5
Leukemia Instances: 73 Features: 1868	MIFS	176.5
	MRMR	176.3
	JMI	278.1
	ReliefF	1.1
	QMIFS-naive	934.4
	QMIFS-ICD	90.6
	QMIFS-GPU	22.5
	QMIFS-MIC	31.8

The results from Tables 5.1 and 5.2 show that our optimizations fare well against the other information-theoretic methods in the classification problem domain. QMIFS-ICD is the fastest in most data sets, especially on the smaller ones. We note that ReliefF is a very fast feature selection method, reason being its low computational complexity [77] and the highly optimized implementation in Matlab. On the other hand it fared quite poorly in the experiments from Chapter 4 with regards to CA, AUC, and Y-index. QMIFS only benefits from parallel GPU and MIC implementations when the number of candidate features in the data set is large, like in the Leukemia data set. The naive implementation of QMIFS is slower in all cases even if the data sets are small, like in the case of the Wine data set. Looking at MIFS, MRMR, and JMI, we can see expected behaviour – the latter being slower than the first two due to its second-order nature.

Table 5.3

Regression problem domain: execution times for selecting half of the features present in the data set for various methods and QMIFS optimizations – first half of the data sets.

<i>Data set</i>	<i>Method</i>	<i>Execution times [s]</i>
Communities Instances: 1993 Features: 100	MIFS	17.4
	MRMR	17.8
	JMI	27.1
	ReliefF	4.5
	QMIFS-naive	1856.9
	QMIFS-ICD	28.3
	QMIFS-GPU	41.7
	QMIFS-MIC	58.4
Parkinson Instances: 1000 Features: 16	MIFS	1.4
	MRMR	1.6
	JMI	1.5
	ReliefF	0.7
	QMIFS-naive	13.8
	QMIFS-ICD	0.8
	QMIFS-GPU	2.6
	QMIFS-MIC	3.3
Wine Quality Instances: 1599 Features: 11	MIFS	1.3
	MRMR	1.4
	JMI	1.4
	ReliefF	0.9
	QMIFS-naive	16.2
	QMIFS-ICD	0.4
	QMIFS-GPU	5.5
	QMIFS-MIC	6.1
Housing Instances: 506 Features: 13	MIFS	0.9
	MRMR	0.9
	JMI	0.9
	ReliefF	0.5
	QMIFS-naive	2.4
	QMIFS-ICD	0.5
	QMIFS-GPU	0.9
	QMIFS-MIC	1.1

Table 5.4

Regression problem domain: execution times for selecting half of the features present in the data set for various methods and QMIFS optimizations – second half of the data sets.

<i>Data set</i>	<i>Method</i>	<i>Execution times [s]</i>
Web	MIFS	3.6
Advertisement	MRMR	3.5
Instances: 950	JMI	6.3
Features: 46	QMIFS	82.4
	ReliefF	0.8
	QMIFS-naive	78.5
	QMIFS-ICD	6.1
	QMIFS-GPU	5.1
	QMIFS-MIC	7.2
Facebook	MIFS	37.5
Instances: 10000	MRMR	38.1
Features: 53	JMI	48.1
	ReliefF	34.4
	QMIFS-naive	105235.6
	QMIFS-ICD	4156.4
	QMIFS-GPU	525.3
	QMIFS-MIC	735.1
Blog	MIFS	214.1
Instances: 5000	MRMR	213.8
Features: 280	JMI	358.2
	ReliefF	58.4
	QMIFS-naive	115643.8
	QMIFS-ICD	4206.3
	QMIFS-GPU	1980.4
	QMIFS-MIC	2752.2
Bank	MIFS	26.9
Instances: 4500	MRMR	26.5
Features: 32	JMI	31.8
	ReliefF	5.5
	QMIFS-naive	1450.3
	QMIFS-ICD	15.7
	QMIFS-GPU	67.5
	QMIFS-MIC	96.3

On the regression problem domain we see that the ICD optimization speeds up QMIFS considerably and is in some cases more effective than the other two implementations namely GPU and MIC. However, on the larger two of the data sets (Facebook and Blog) the latter two are considerably faster than ICD. The naive implementation is much slower on the largest data sets even infeasible to use. The other information-theoretic measures are in quite a few cases faster than any implementation of QMIFS, probably due to the fact that the discretization performed on the data was simple equal frequency binning, which saves time in comparison to MDL. ReliefF proves to be the fastest, but again it does not fare so well in terms of improvement of RMSE as can be seen from the previous chapter. It still did achieve a better rank than JMI and MIFS while also being much faster on the larger data sets.

To provide a more thorough insight into the effectiveness of the optimized variants of QMIFS we perform a second batch of experiments on artificial data sets with continuous features and output. The data was generated randomly using uniform distribution. The experimental set-up is the same as in the previous case. Here we exclude the other methods from the analysis in order to put the emphasis on the QMIFS optimizations. We vary the number of instances, the number of features, and the number of selected features. Additionally we compute the speed-up (S) of QMIFS-ICD, QMIFS-GPU, and QMIFS-MIC against QMIFS-naive. The speed-up is defined as the quotient between execution time of QMIFS-naive and the execution time of QMIFS-ICD, QMIFS-GPU, or QMIFS-MIC

$$S = \frac{T_{QMIFS-naive}}{T_{method}}. \quad (5.23)$$

Each measurement was performed ten times with the execution times averaged to obtain the final result. The relative error of the measurements is around 2% in all cases. The first batch of results in Tab. 5.5 shows how the number of instances affects the execution times for each variant of QMIFS. We set the number of features in the data set to 300 and instructed the algorithm to select half of the features (150) from the data set.

Table 5.5

Execution times of different variants of QMIFS with respect to the number of instances.

Instances	QMIFS-naive	QMIFS-ICD	S	QMIFS-GPU	S	QMIFS-MIC	S
	Time [s]	Time [s]		Time [s]		Time [s]	
100	$3.72 \cdot 10^1$	$6.10 \cdot 10^0$	6	$5.00 \cdot 10^{-1}$	78	$6.61 \cdot 10^{-1}$	57
325	$3.71 \cdot 10^2$	$2.59 \cdot 10^1$	14	$4.27 \cdot 10^0$	87	$6.01 \cdot 10^0$	62
550	$1.10 \cdot 10^3$	$5.02 \cdot 10^1$	22	$1.22 \cdot 10^1$	90	$1.73 \cdot 10^1$	64
775	$2.25 \cdot 10^3$	$7.21 \cdot 10^1$	31	$2.43 \cdot 10^1$	93	$3.45 \cdot 10^1$	65
1000	$3.80 \cdot 10^3$	$9.37 \cdot 10^1$	41	$4.15 \cdot 10^1$	92	$6.02 \cdot 10^1$	63

We can see that QMIFS-naive is the slowest in all case by a considerable margin. The execution time grows quadratically, which is the expected behaviour. QMIFS-ICD is considerably faster with speed-ups reaching up to 40. The execution time grows approximately linearly due to the efficient ICD decomposition of the data. QMIFS-GPU is the fastest with peak speed-up reaching 93. Even though it shares the quadratic growth of execution times with QMIFS-naive, the raw computational capabilities of the GPU still make it considerably faster than QMIFS-ICD in our test scenario. The MIC implementation is somewhere in between peaking at a speed-up of around 65. This can be attributed to the lower raw computational power of the Xeon Phi coprocessor in comparison to the K20 GPU.

In the next set of results presented in Tab. 5.6 we show how the number of features in the data set affects the execution times of QMIFS implementations. We vary the number of features, set the number of instances to 550, and perform selection of half of the total number of features. Again the data are random and uniformly distributed.

Table 5.6

Execution times of different variants of QMIFS with respect to the number of features.

Features	QMIFS-naive	QMIFS-ICD	S	QMIFS-GPU	S	QMIFS-MIC	S
	Time [s]	Time [s]		Time [s]		Time [s]	
100	$1.27 \cdot 10^2$	$5.31 \cdot 10^0$	24	$1.91 \cdot 10^0$	67	$2.86 \cdot 10^0$	45
200	$5.14 \cdot 10^2$	$2.21 \cdot 10^1$	23	$5.91 \cdot 10^0$	87	$8.40 \cdot 10^0$	61
300	$1.10 \cdot 10^3$	$4.95 \cdot 10^1$	22	$1.22 \cdot 10^1$	90	$1.73 \cdot 10^1$	63
400	$1.97 \cdot 10^3$	$8.91 \cdot 10^1$	22	$1.85 \cdot 10^1$	107	$2.84 \cdot 10^1$	69
500	$3.05 \cdot 10^3$	$1.36 \cdot 10^2$	22	$2.81 \cdot 10^1$	109	$4.34 \cdot 10^1$	70

This scenario shows similar behaviour to the previous. The execution time grows quadratically with respect to the number of features in all three cases. This is unsurprising, as the number of evaluations of the criterion function grows quadratically with respect to the number of features. Again QMIFS-ICD, QMIFS-GPU, and QMIFS-MIC are much faster than the naive implementation. The speed-up of QMIFS-ICD is approximately constant (23) across different test cases, since ICD only reduces the computational complexity of criterion function evaluation. QMIFS-GPU is the fastest, with speed-up increasing as the number of features grows, peaking at 109 for the largest data set. Such behaviour is expected, since increasing the number of feature also increases the level of parallelism available in the algorithm. The behaviour of QMIFS-MIC is similar to QMIFS-GPU due to the same approach to parallelization it achieves a peek speed-up of 70.

The third set of results in Tab. 5.7 deals with how the number of required features affects the execution times. We set the number of features to 200 and the number of instances to 550 and performed selection of 25%, 50%, and 75% of features. The data are random and uniformly distributed.

Table 5.7

Execution times of different variants of QMIFS with respect to the number of selected features.

selected features	QMIFS-naive	QMIFS-ICD		QMIFS-GPU		QMIFS-MIC	
	Time [s]	Time [s]	S	Time [s]	S	Time [s]	S
50	$2.95 \cdot 10^2$	$1.30 \cdot 10^1$	23	$5.91 \cdot 10^0$	50	$8.40 \cdot 10^0$	35
100	$5.14 \cdot 10^2$	$2.21 \cdot 10^1$	23	$5.91 \cdot 10^0$	87	$8.40 \cdot 10^0$	61
150	$6.25 \cdot 10^2$	$2.74 \cdot 10^1$	23	$5.91 \cdot 10^0$	106	$8.40 \cdot 10^0$	74

Here the execution times grow linearly in the case of QMIFS-naive and QMIFS-ICD. The latter is around 23 times faster due to the more efficient computation of the criterion function. QMIFS-GPU and QMIF-MIC behave differently, with constant execution time, since the bulk of the work is done before performing actual feature selection and is the same regardless of the number of features we want to select. The GPU implementation is again the fastest with speed-ups of up to 106 and MIC second with a speed-up of 74.

Next experiment addresses the question on how the type of data affects the efficiency of incomplete Cholesky distribution (see Chapter 5.1). Table 5.8 compares the execu-

tion times of QMIFS-ICD on two types of data sets. In the first case we use uniform distribution to generate the values and in the second case we use normal distribution. The experimental set-up regarding number of features, number of instances, and number of selected features is the same as in the first experiment. The execution times for QMIFS-naive and QMIFS-GPU are the same as in Tab. 5.5, since the way data are generated does not affect them.

Table 5.8

Execution times of QMIFS utilizing ICD on data generated using uniform and random distribution with respect to the number of instances. The execution times are given in seconds.

<i>Instances</i>	QMIFS-naive	QMIFS-ICD	
		Uniform distribution [s]	Normal distribution [s]
100	$3.72 \cdot 10^1$	$6.10 \cdot 10^0$	$8.82 \cdot 10^0$
325	$3.71 \cdot 10^2$	$2.59 \cdot 10^1$	$5.04 \cdot 10^1$
550	$1.10 \cdot 10^3$	$5.02 \cdot 10^1$	$1.17 \cdot 10^2$
775	$2.25 \cdot 10^3$	$7.21 \cdot 10^1$	$1.48 \cdot 10^2$
1000	$3.80 \cdot 10^3$	$9.37 \cdot 10^1$	$2.15 \cdot 10^2$

We can clearly see that QMIFS-ICD is much slower when the features' values are normally distributed. Obviously the Incomplete Cholesky decomposition produces larger matrices in case of normally distributed data, which causes longer execution times. Nevertheless, even in the case of normally distributed data, QMIFS-ICD is still much faster than QMIFS-naive.

These results clearly show that an efficient implementation of QMIFS on the GPU is possible. It offers considerable time savings, which can be crucial when dealing with large data sets. Our GPU implementation, although having higher computational complexity, is insensitive to the type of data we are analyzing and is in this regard advantageous to the QMIFS-naive. The MIC implementation exhibits less gain in comparison to ICD, mostly due to the less capable hardware of the Xeon Phi. It can still be useful when dealing with large data sets in order to complement the GPU implementation.

The last experiment tries to address the feasibility of the heterogeneous QMIFS implementation. To see if it provides considerable time savings we select the Blog data set, which takes the most time to analyze using QMIFS and try to employ a combina-

tion of QMIFS-naive, QMIFS-GPU, and QMIFS-MIC and run it on all 12 cores of the Xeon processor, one K20, and one Xeon Phi 5110P. In order to distribute work between them we used the execution time measurements of different implementations on the Blog data set found in Table 5.4 – this reduces overhead and improves utilization of the hardware. Each implementation, thus receives the amount of work proportional to its computational performance. This means that the GPU gets assigned 60% of the work, CPU 12%, and Xeon Phi 28%. Using this configuration we reduce the computational time to 1356 seconds. A speed-up of 1.46 in comparison to only QMIFS-GPU. Of course having more hardware at hand could improve this result even more.

Conclusion

6

In the thesis we focus on information-theoretic approach to feature selection. We review the foundations of information theory and look into its extensions established by Renyi. We investigate the field of feature selection and its close relationship with information theory. Based on the previous research in this field, we establish the deficiencies of existing methods and propose a quadratic mutual information feature selection method (QMIFS). Our goal was to detect second-order non-linear relations between features and the output, similarly to joint mutual information. Additionally, we focus on the analysis of both discrete and continuous features and outputs, avoiding the intermediate step of estimating underlying probability density functions using histograms or kernel density estimation. To achieve these goals, we employ a quadratic mutual information (QMI) measure, as it enables direct estimation from the data samples. The measure itself does not exhibit all the properties intrinsic to mutual information measure, and additional effort was needed to compensate for deficiencies.

We compare our method to three representative methods based on information-theoretic measures: mutual information feature selection (MIFS), minimum redundancy maximum relevance (MRMR), and joint mutual information (JMI). Additionally, we include ReliefF in comparison as one of the most commonly used, non-information theoretic feature selection methods. The methods are compared indirectly, on the classification problem domain using models built by the classification tree learning machine, and on the regression problem domain using the regression tree learning machine. The results show that our method offers similar performance on the classification problem domain in terms of classification accuracy, area under the curve, and Youden index as the other three. When dealing with regression, it compares favourably to the others regarding root-mean-squared-error.

To improve the computational complexity of our method, we adopt the incomplete Cholesky factorization (ICD) in the computation of quadratic mutual information. We extend the original proposal of using ICD in the computation of QMI to the three-variate QMI.

Recent trends in machine learning favour the usage of graphics processing units (GPUs) and coprocessors to tackle complex learning processes and analysis of big data sets. We design QMIFS with this in mind and adopt it for execution on the GPU using CUDA programming model and Many integrated core architecture (MIC) using OpenMP. Furthermore, we develop a proof of concept implementation of QMIFS, which distributes work to all available processing resources (CPUs, GPUs, and MIC

coprocessors). We achieve considerable improvement of the execution time, making QMIFS suitable for usage on large data sets.

We conclude that our method is universal, capable of feature selection on classification or regression problem domain. QMIFS does not need an additional preprocessing step to estimate the probability density function, as is the case in the other three methods. This and the fact that it avoids using parameters makes it simple to use for non-experts in the field. Experiments show that straightforward estimation of QMI from data samples using quadratic Renyi entropy and Gaussian kernels does a good job at identifying the important information in the data. Through the usage of ICD, it offers considerable execution time savings compared to other feature selection methods coupled with advanced discretization techniques. Furthermore, the GPU and MIC implementations make it feasible for usage on very large data sets.

6.1 Future work

Our work opens multiple fronts for future research. One of the deficiencies of quadratic mutual information estimator is its sensitivity to kernel width selection. Other approaches towards estimating the kernel width could be investigated in order to improve the performance of QMIFS.

The literature on extensions of information-theory proposes a multitude of dependency measures. Our approach to feature selection can be modified to use other evaluation criteria. So, a possible direction for future research is the investigation of other information-theoretic measures for compatibility with our approach to increase their potential.

Our GPU and MIC implementations avoid the incomplete Cholesky approach because of its memory requirements and additional management, but could prove useful when dealing with data sets that include a lot feature instances. The problem lies also in the optimization of the algorithm for these two architectures. Consequently further work could go towards adopting it for execution on the GPU and/or MIC.



Supplement to chapter
“Proposed method”

A

Table A.1

Classification problem domain: values of measures for 3 selected features for the first half of the data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

Data set	Method	Measure		
		CA	AUC	Y-index
Chess	MIFS	0.898	0.932	0.790
	MRMR	0.899	0.933	0.791
	JMI	0.899	0.933	0.791
	QMIFS	0.899	0.933	0.791
	ReliefF	0.899	0.932	0.791
	All features	0.969	0.985	0.942
	SE	0.0004	0.0003	0.0009
Breast cancer	MIFS	0.928	0.942	0.857
	MRMR	0.931	0.946	0.869
	JMI	0.926	0.947	0.860
	QMIFS	0.931	0.948	0.866
	ReliefF	0.719	0.743	0.436
	All features	0.922	0.926	0.848
	SE	0.0006	0.0008	0.001
Ionosphere	MIFS	0.866	0.881	0.727
	MRMR	0.869	0.883	0.737
	JMI	0.900	0.918	0.801
	QMIFS	0.876	0.901	0.753
	ReliefF	0.816	0.832	0.628
	All features	0.878	0.881	0.758
	SE	0.0011	0.0013	0.0023
Sonar	MIFS	0.685	0.733	0.450
	MRMR	0.679	0.725	0.437
	JMI	0.714	0.736	0.478
	QMIFS	0.691	0.737	0.444
	ReliefF	0.648	0.687	0.363
	All features	0.714	0.724	0.460
	SE	0.0018	0.0021	0.0032

Table A.2

Classification problem domain: values of measures for 3 selected features for the second half of data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Wine	MIFS	0.908	0.908	0.815
	MRMR	0.883	0.903	0.777
	JMI	0.883	0.905	0.779
	QMIFS	0.884	0.905	0.779
	ReliefF	0.873	0.895	0.769
	All features	0.906	0.912	0.813
	SE	0.0014	0.0025	0.0030
Cervical cancer	MIFS	0.842	0.861	0.692
	MRMR	0.846	0.866	0.708
	JMI	0.879	0.918	0.762
	QMIFS	0.805	0.814	0.627
	ReliefF	0.622	0.666	0.305
	All features	0.761	0.785	0.536
	SE	0.0033	0.0037	0.0066
Spine	MIFS	0.781	0.821	0.561
	MRMR	0.782	0.821	0.561
	JMI	0.766	0.805	0.552
	QMIFS	0.811	0.850	0.604
	ReliefF	0.780	0.819	0.558
	All features	0.788	0.791	0.566
	SE	0.0012	0.0021	0.0035
Leukemia	MIFS	0.927	0.082	0.006
	MRMR	0.928	0.079	0.002
	JMI	0.924	0.083	0.000
	QMIFS	0.919	0.070	0.003
	ReliefF	0.908	0.083	0.000
	All features	0.865	0.147	0.000
	SE	0.0018	0.0023	0.0010

Table A.3

Classification problem domain: values of measures for 5 selected features for the first half of the data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Chess	MIFS	0.935	0.957	0.866
	MRMR	0.934	0.957	0.866
	JMI	0.934	0.957	0.867
	QMIFS	0.942	0.977	0.887
	ReliefF	0.895	0.934	0.792
	All features	0.969	0.985	0.942
	SE	0.0004	0.0003	0.0008
Breast cancer	MIFS	0.930	0.941	0.859
	MRMR	0.930	0.942	0.863
	JMI	0.935	0.947	0.874
	QMIFS	0.926	0.943	0.857
	ReliefF	0.928	0.939	0.856
	All features	0.922	0.926	0.848
	SE	0.0006	0.0008	0.0013
Ionosphere	MIFS	0.893	0.901	0.786
	MRMR	0.888	0.910	0.779
	JMI	0.899	0.921	0.803
	QMIFS	0.881	0.899	0.765
	ReliefF	0.797	0.808	0.594
	All features	0.878	0.881	0.758
	SE	0.0012	0.0014	0.0024
Sonar	MIFS	0.714	0.745	0.486
	MRMR	0.686	0.720	0.432
	JMI	0.707	0.734	0.470
	QMIFS	0.693	0.729	0.445
	ReliefF	0.633	0.660	0.331
	All features	0.714	0.724	0.460
	SE	0.0017	0.0020	0.0033

Table A.4

Classification problem domain: values of measures for 5 selected features for the second half of data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Wine	MIFS	0.913	0.919	0.829
	MRMR	0.909	0.916	0.826
	JMI	0.910	0.918	0.828
	QMIFS	0.918	0.927	0.846
	ReliefF	0.917	0.913	0.807
	All features	0.906	0.912	0.813
	SE	0.0014	0.0027	0.0030
Cervical cancer	MIFS	0.826	0.861	0.669
	MRMR	0.850	0.863	0.710
	JMI	0.849	0.877	0.711
	QMIFS	0.770	0.779	0.570
	ReliefF	0.774	0.783	0.572
	All features	0.761	0.785	0.536
	SE	0.0031	0.0034	0.0066
Spine	MIFS	0.770	0.791	0.537
	MRMR	0.802	0.841	0.596
	JMI	0.801	0.831	0.588
	QMIFS	0.800	0.837	0.587
	ReliefF	0.803	0.842	0.596
	All features	0.788	0.791	0.566
	SE	0.0013	0.0021	0.0034
Leukemia	MIFS	0.927	0.080	0.004
	MRMR	0.913	0.094	0.000
	JMI	0.920	0.089	0.000
	QMIFS	0.923	0.067	0.004
	ReliefF	0.914	0.091	0.000
	All features	0.865	0.147	0.000
	SE	0.0019	0.0021	0.0012

Table A.5

Classification problem domain: values of measures for 7 selected features for the first half of the data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Chess	MIFS	0.934	0.956	0.865
	MRMR	0.932	0.964	0.866
	JMI	0.933	0.965	0.866
	QMIFS	0.938	0.976	0.886
	ReliefF	0.935	0.957	0.869
	All features	0.969	0.985	0.942
	SE	0.0004	0.0003	0.0007
Breast cancer	MIFS	0.929	0.939	0.858
	MRMR	0.932	0.942	0.868
	JMI	0.935	0.944	0.873
	QMIFS	0.932	0.941	0.866
	ReliefF	0.932	0.941	0.863
	All features	0.922	0.926	0.848
	SE	0.0006	0.0008	0.001
Ionosphere	MIFS	0.897	0.904	0.794
	MRMR	0.888	0.903	0.780
	JMI	0.897	0.915	0.796
	QMIFS	0.881	0.897	0.764
	ReliefF	0.891	0.907	0.788
	All features	0.878	0.881	0.758
	SE	0.001	0.001	0.002
Sonar	MIFS	0.728	0.755	0.504
	MRMR	0.681	0.711	0.424
	JMI	0.731	0.759	0.510
	QMIFS	0.734	0.759	0.506
	ReliefF	0.674	0.702	0.407
	All features	0.714	0.724	0.460
	SE	0.002	0.002	0.003

Table A.6

Classification problem domain: values of measures for 7 selected features for the second half of data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Wine	MIFS	0.911	0.919	0.826
	MRMR	0.910	0.917	0.828
	JMI	0.911	0.916	0.829
	QMIFS	0.915	0.923	0.839
	ReliefF	0.913	0.910	0.813
	All features	0.906	0.912	0.813
	SE	0.0014	0.0017	0.0030
Cervical cancer	MIFS	0.828	0.858	0.669
	MRMR	0.848	0.862	0.708
	JMI	0.851	0.879	0.716
	QMIFS	0.803	0.824	0.638
	ReliefF	0.753	0.772	0.525
	All features	0.761	0.785	0.536
	SE	0.003	0.003	0.007
Spine	MIFS	0.766	0.769	0.527
	MRMR	0.797	0.821	0.589
	JMI	0.797	0.831	0.586
	QMIFS	0.798	0.831	0.589
	ReliefF	0.804	0.834	0.597
	All features	0.788	0.791	0.566
	SE	0.001	0.002	0.003
Leukemia	MIFS	0.928	0.079	0.003
	MRMR	0.915	0.091	0.000
	JMI	0.925	0.065	0.003
	QMIFS	0.916	0.093	0.000
	ReliefF	0.913	0.095	0.000
	All features	0.865	0.147	0.000
	SE	0.003	0.002	0.001

Table A.7

Classification problem domain: values of measures for 10 selected features for the first half of the data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Chess	MIFS	0.933	0.955	0.864
	MRMR	0.945	0.981	0.891
	JMI	0.943	0.983	0.890
	QMIFS	0.945	0.982	0.903
	ReliefF	0.935	0.976	0.879
	All features	0.969	0.985	0.942
	SE	0.0004	0.0003	0.0007
Breast cancer	MIFS	0.931	0.936	0.861
	MRMR	0.929	0.938	0.862
	JMI	0.931	0.94	0.866
	QMIFS	0.931	0.94	0.865
	ReliefF	0.929	0.935	0.861
	All features	0.922	0.926	0.848
	SE	0.0006	0.0008	0.0013
Ionosphere	MIFS	0.893	0.902	0.789
	MRMR	0.887	0.898	0.776
	JMI	0.907	0.919	0.819
	QMIFS	0.878	0.893	0.759
	ReliefF	0.909	0.921	0.823
	All features	0.878	0.881	0.758
	SE	0.0010	0.0013	0.0020
Sonar	MIFS	0.716	0.737	0.487
	MRMR	0.738	0.761	0.518
	JMI	0.737	0.760	0.521
	QMIFS	0.722	0.740	0.481
	ReliefF	0.713	0.736	0.470
	All features	0.714	0.724	0.460
	SE	0.0019	0.0022	0.0034

Table A.8

Classification problem domain: values of measures for 10 selected features for the second half of data sets. For comparison the values obtained with all features are shown. The last row at each data set lists the maximum standard errors (SE) of the measures

<i>Data set</i>	<i>Method</i>	<i>Measure</i>		
		<i>CA</i>	<i>AUC</i>	<i>Y-index</i>
Wine	MIFS	0.916	0.920	0.835
	MRMR	0.907	0.915	0.824
	JMI	0.909	0.915	0.826
	QMIFS	0.916	0.924	0.842
	ReliefF	0.907	0.911	0.817
	All features	0.906	0.912	0.813
	SE	0.0014	0.0016	0.0030
Cervical cancer	MIFS	0.821	0.856	0.660
	MRMR	0.828	0.845	0.670
	JMI	0.852	0.874	0.715
	QMIFS	0.788	0.811	0.619
	ReliefF	0.773	0.807	0.576
	All features	0.761	0.785	0.536
	SE	0.0033	0.0034	0.0066
Spine	MIFS	0.792	0.799	0.578
	MRMR	0.791	0.798	0.574
	JMI	0.791	0.803	0.570
	QMIFS	0.792	0.806	0.573
	ReliefF	0.793	0.803	0.576
	All features	0.788	0.791	0.566
	SE	0.0012	0.0021	0.0029
Leukemia	MIFS	0.929	0.077	0.003
	MRMR	0.914	0.094	0.000
	JMI	0.914	0.095	0.000
	QMIFS	0.914	0.086	0.006
	ReliefF	0.916	0.090	0.000
	All features	0.865	0.147	0.000
	SE	0.0018	0.0023	0.0015

Table A.9

Regression problem domain: values of RMSE measure for 3 selected features for the first half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Communities	MIFS	0.176
	MRMR	0.176
	JMI	0.182
	QMIFS	0.190
	ReliefF	0.206
	All features	0.190
	SE	0.0002
Parkinson telemonitoring	MIFS	9.116
	MRMR	9.106
	JMI	8.399
	QMIFS	8.827
	ReliefF	8.387
	All features	8.300
	SE	0.0132
Wine quality	MIFS	0.772
	MRMR	0.760
	JMI	0.798
	QMIFS	0.760
	ReliefF	0.785
	All features	0.771
	SE	0.0010
Housing	MIFS	5.489
	MRMR	5.083
	JMI	5.076
	QMIFS	5.248
	ReliefF	4.751
	All features	4.634
	SE	0.0223

Table A.10

Regression problem domain: values of RMSE measure for 3 selected features for the second half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Web advertisement	MIFS	3.746
	MRMR	3.746
	JMI	3.562
	QMIFS	2.953
	ReliefF	3.792
	All features	3.590
	SE	0.0063
Facebook	MIFS	32.230
	MRMR	24.655
	JMI	31.873
	QMIFS	24.841
	ReliefF	32.152
	All features	26.912
	SE	0.1362
Blog	MIFS	38.604
	MRMR	42.402
	JMI	38.078
	QMIFS	33.380
	ReliefF	39.351
	All features	37.896
	SE	0.2684
Bank	MIFS	0.114
	MRMR	0.113
	JMI	0.113
	QMIFS	0.113
	ReliefF	0.113
	All features	0.117
	SE	0.0001

Table A.11

Regression problem domain: values of RMSE measure for 5 selected features for the first half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Communities	MIFS	0.180
	MRMR	0.180
	JMI	0.181
	QMIFS	0.190
	ReliefF	0.190
	All features	0.190
	SE	0.0002
Parkinson telemonitoring	MIFS	8.890
	MRMR	8.918
	JMI	8.324
	QMIFS	8.336
	ReliefF	8.240
	All features	8.300
	SE	0.0132
Wine quality	MIFS	0.789
	MRMR	0.758
	JMI	0.775
	QMIFS	0.774
	ReliefF	0.775
	All features	0.771
	SE	0.0010
Housing	MIFS	5.116
	MRMR	5.104
	JMI	5.027
	QMIFS	4.992
	ReliefF	4.602
	All features	4.634
	SE	0.0237

Table A.12

Regression problem domain: values of RMSE measure for 5 selected features for the second half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Web advertisement	MIFS	3.757
	MRMR	3.755
	JMI	3.699
	QMIFS	2.956
	ReliefF	3.748
	All features	3.590
	SE	0.0063
Facebook	MIFS	31.918
	MRMR	24.660
	JMI	31.807
	QMIFS	25.284
	ReliefF	25.107
	All features	26.912
	SE	0.1362
Blog	MIFS	35.675
	MRMR	38.326
	JMI	39.730
	QMIFS	33.943
	ReliefF	40.172
	All features	37.896
	SE	0.2725
Bank	MIFS	0.111
	MRMR	0.111
	JMI	0.111
	QMIFS	0.111
	ReliefF	0.115
	All features	0.117
	SE	0.0001

Table A.13

Regression problem domain: values of RMSE measure for 7 selected features for the first half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Communities	MIFS	0.186
	MRMR	0.180
	JMI	0.183
	QMIFS	0.183
	ReliefF	0.191
	All features	0.190
	SE	0.0003
Parkinson telemonitoring	MIFS	8.93
	MRMR	8.23
	JMI	8.43
	QMIFS	8.42
	ReliefF	8.32
	All features	8.30
	SE	0.02
Wine quality	MIFS	0.772
	MRMR	0.773
	JMI	0.770
	QMIFS	0.771
	ReliefF	0.778
	All features	0.771
	SE	0.001
Housing	MIFS	5.12
	MRMR	5.06
	JMI	4.85
	QMIFS	4.61
	ReliefF	4.49
	All features	4.63
	SE	0.03

Table A.14

Regression problem domain: values of RMSE measure for 7 selected features for the second half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Web advertisement	MIFS	3.668
	MRMR	3.703
	JMI	3.567
	QMIFS	3.174
	ReliefF	3.589
	All features	3.590
	SE	0.006
Facebook	MIFS	26.267
	MRMR	24.655
	JMI	25.750
	QMIFS	32.230
	ReliefF	25.447
	All features	26.912
	SE	0.136
Blog	MIFS	38.181
	MRMR	38.886
	JMI	41.266
	QMIFS	34.176
	ReliefF	39.577
	All features	37.896
	SE	0.287
Bank	MIFS	0.111
	MRMR	0.111
	JMI	0.111
	QMIFS	0.112
	ReliefF	0.116
	All features	0.117
	SE	0.001

Table A.15

Regression problem domain: values of RMSE measure for 10 selected features for the first half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Communities	MIFS	0.188
	MRMR	0.181
	JMI	0.183
	QMIFS	0.185
	ReliefF	0.187
	All features	0.190
	SE	0.0002
Parkinson telemonitoring	MIFS	8.212
	MRMR	8.229
	JMI	8.475
	QMIFS	8.447
	ReliefF	8.334
	All features	8.300
	SE	0.0139
Wine quality	MIFS	0.773
	MRMR	0.773
	JMI	0.772
	QMIFS	0.770
	ReliefF	0.776
	All features	0.771
	SE	0.0010
Housing	MIFS	4.742
	MRMR	4.930
	JMI	4.896
	QMIFS	4.542
	ReliefF	4.487
	All features	4.634
	SE	0.0236

Table A.16

Regression problem domain: values of RMSE measure for 10 selected features for the second half of the data sets. Column *All features* holds the value of RMSE obtained with all features. The maximum standard error (SE) is given in the last row for each dataset.

<i>Data set</i>	<i>Method</i>	<i>RMSE</i>
Web advertisement	MIFS	3.466
	MRMR	3.526
	JMI	3.582
	QMIFS	3.175
	ReliefF	3.552
	All features	3.590
	SE	0.0063
Facebook	MIFS	25.456
	MRMR	24.986
	JMI	32.344
	QMIFS	25.741
	ReliefF	25.777
	All features	26.912
	SE	0.1362
Blog	MIFS	38.279
	MRMR	34.679
	JMI	39.200
	QMIFS	34.570
	ReliefF	39.822
	All features	37.896
	SE	0.2714
Bank	MIFS	0.113
	MRMR	0.112
	JMI	0.112
	QMIFS	0.112
	ReliefF	0.115
	All features	0.117
	SE	0.0001



Razširjeni povzetek

B

B.1 Uvod

Visoko-dimenzionalne podatkovne zbirke najdemo na veliko področjih znanosti, kot so procesiranje slik in signalov, analiza bioloških in medicinskih podatkov ter oglaševanja. Zbiranje čim večje količine značilk v teoriji samo poveča količino informacij o opazovanem sistemu. Težava nastane pri analizi takih podatkovnih zbirk, saj veliko orodij za analizo podatkov pri veliki količini značilk preprosto odpove [1].

Reševanja te problematike se lahko lotimo na dva načina. Ena možnost je razvoj sodobnih metod, ki so sposobne modelirati visoko-dimenzionalne podatke v nizko-dimenzionalnem prostoru. Druga možnost je zmanjšanje dimenzionalnosti podatkovnih zbirk, ne da bi pri tem izgubili pomembne informacije. Tu sta poznana dva pristopa: gradnja značilk, kjer izvirne značilke združujemo/preoblikujemo in s tem dobimo nov nabor značilk, in iskanje značilk, kjer med izvirnimi značilkami želimo poiskati najbolj pomembne oziroma informativne. Slednji pristop ohrani izvirne značilke in je zato lahko v pomoč pri interpretaciji rezultatov analize.

Pri iskanju pomembnih značilk se moramo nasloniti na neko kriterijsko funkcijo, ki oceni pomembnost posamezne značilke. Možnosti je veliko in segajo od raznih razdalj in divergenc do informacijskih mer [2].

Informacijske mere so zagotovo med najbolj zanimivimi, saj izhajajo iz same definicije izbiranja značilk: iskanje nabora značilk, ki nam da največ informacije o opazovanem sistemu. Med pogosto uporabljene informacijske mere spadajo medsebojna informacija, skupaj z raznimi dopolnitvami, in relativna entropija [4]. Možne kandidatke so tudi mere osnovane na posplošitvah Shannonove entropije, kot na primer Renyijeve [5] in Tsallisove [6] entropije. Vse informacijske mere slonijo na računanju verjetnostnih porazdelitev podatkov s pomočjo histogramov ali jedrnih funkcij. Ta pristop je lahko problematičen iz dveh razlogov: računske zahtevnosti in točnosti ocene verjetnostne porazdelitve, še posebej, ko imamo opravka z visoko-dimenzionalnimi verjetnostnimi porazdelitvami.

Večina informacijsko-teoretičnih metod za izbiro značilk lahko preučuje le diskretne značilke. Če podatkovna zbirka vključuje tudi zvezne značilke, je te potrebno diskretizirati, kar lahko dodatno zaplete postopek izbire značilk. Temu se lahko izognemo, če uporabimo katero izmed metod, ki informacijske-mere oceni neposredno iz podatkov [20]. Eno izmed rešitev so v svojem delu predstavili Principe in sod. [7], kjer so z uporabo Renyijeve entropije in metode Parzenovih oken razvili način za učinkovit

izračun kvadratne medsebojne informacije neposredno iz podatkov bodisi diskretnih ali zveznih.

Izvajalni čas metod za izbiro značilk postane problematičen, ko imamo opraviti z zelo velikimi podatkovnimi zbirkami. Računska moč masovno-paralelnih arhitektur, zgled katerih so grafične procesne enote [8], omogoča, da dosežemo razumne čase izvajanja ter s tem omogočimo analizo velikih podatkovnih zbirk, kot so na primer genetski podatki [10].

B.1.1 Prispevki k znanosti

Glavna tema disertacije je uvedba nove informacijsko-teoretične mere in metode za iskanje pomembnih značilk v podatkih. Primarni cilj je odkrivanje interakcij med pari značilk, ki lahko v določenih primerih prispevajo k jasnejši interpretaciji opazovanih podatkov. Povzetek prispevkov k znanosti:

- *Uvedba nove metode za izbiro značilk, sloneče na posplošenih merah informacije.* Predlagamo algoritem QMIFS (ang. quadratic mutual information feature selection), ki kot kriterijsko funkcijo uporablja kvadratno medsebojno informacijo. Predlagana metoda uporablja požrešno metodo preiskovanja prostora značilk in lahko med značilkami in izhodom zazna odvisnosti drugega reda. Zmore analizo tako diskretnih kot tudi zveznih značilk, kar odpravlja potrebo po predprocesiranju. Prav tako ne potrebuje nastavljanja parametrov, ki lahko laikom na področju povzročajo težave. Uporabljamo jo lahko tako pri problemih uvrščanja kot tudi pri regresijskih problemih, kjer pripomore k izboljšanju podatkovnega modela.
- *Optimizacija računanja kriterijske funkcije s pomočjo nepopolne dekompozicije Choleskega.* V kriterijski funkciji metode QMIFS smo uporabili nepopolno dekompozicijo Choleskega in s tem zmanjšali računsko zahtevnost v odvisnosti od števila primerov iz kvadratne v linearno. Da smo to dosegli, smo morali izpeljati enačbe za uporabo nepopolne dekompozicije Choleskega pri računanju kvadratne medsebojne informacije treh spremenljivk.
- *Paralelizacija metode za uporabo na masovno-paralelnih sistemih.* Predlagamo modifikacijo metode QMIFS, ki omogoča njeno izvajanje na masovno-paralelnih arhitekturah. Identificirali smo računsko najbolj zahtevne dele algoritma in jih

prilagodili za izvajanje na masovno-paralelnih arhitekturah kot so grafične procesne enote in paralelni računski koprocesorji. Poseben poudarek smo dali zmanjšanju režije pri prenašanju podatkov in koordinacije med centralno procesno enoto ter grafično procesno enoto/koprocesorjem. Poleg tega smo raziskali možnost izkoriščanja vseh računskih zmogljivosti hkrati, kar pripomore še k dodatnemu zmanjšanju časa procesiranja.

B.2 Teorija informacij

Teorijo informacij, kot jo poznamo danes, je postavil Shannon v svojem delu iz leta 1948 [12]. To področje znanosti se ukvarja z vrednotenjem, hranjenjem in prenašanjem informacije. Shannon je v svojem delu informacijo povezal z nedoločenostjo oziroma entropijo, ki jo lahko interpretiramo kot povprečno količino informacije, ki jo moramo izvedeti o neki naključni spremenljivki, da popolnoma odstranimo negotovost o njeni vrednosti. Iz tega sledi, da je informacija neposredno odvisna od števila stanj, ki jih lahko zavzame naključna spremenljivka, in verjetnosti posameznih stanj. Informacijo prav tako kot nedoločenost merimo v bitih. Najmanjša je, ko je vrednost naključne spremenljivke natančno določena, največja pa, ko so vse vrednosti enako verjetne.

V primeru, da hkrati opazujemo več naključnih spremenljivk, ki so lahko med sabo na nek način povezane, nas pogosto zanima, koliko informacije o naključni spremenljivki vsebuje neka druga naključna spremenljivka. To nam pove medsebojna informacija [4]. Ta je nenegativna količina, ki je enaka nič le v primeru, ko sta spremenljivki med sabo neodvisni. V primeru, da ena spremenljivka v popolnosti opiše drugo spremenljivko, pa je medsebojna informacija največja in kar enaka nedoločenosti opazovane naključne spremenljivke. Koncept medsebojne informacije je mogoče posplošiti tudi na več spremenljivk hkrati, vendar je tu možnih več interpretacij. Avtorji so tekom razvoja področja teorije informacij predlagali več načinov merjenja medsebojne informacije med več spremenljivkami hkrati. Med bolj znanimi so popolna korelacija, multivariatna medsebojna informacija in sinergija [14].

Poleg klasične Shannonove definicije nedoločenosti in informacije obstajajo tudi njene posplošitve [16]. Ena izmed najbolj znanih je Renyijeva posplošitev [17]. V svojem delu je Renyi poskušal najti najbolj splošno definicijo informacijske mere, ki še ohranja aditivnost neodvisnih dogodkov in je združljiva z aksiomi Kolmogorova. Predlagal je parametrično družino informacijskih mer – Renyijevih entropij. Shanno-

nova entropija je v njegovi posplošitvi le ena izmed možnih parametričnih interpretacij Renyijeve entropije.

Pri računanju informacijskih mer moramo razlikovati med diskretnimi in zveznimi naključnimi spremenljivkami. V vsakem primeru potrebujemo verjetnostno porazdelitev vrednosti naključne spremenljivke, da lahko pridemo do ocene opazovane informacijske mere. V primeru, da je naključna spremenljivka diskretna, lahko verjetnostno porazdelitev ocenimo s histogramom [18]. Če je naključna spremenljivka zvezna, imamo na razpolago več možnosti. Vrednosti lahko diskretiziramo in tako problem prevedemo v diskretnega. To lahko povzroči dodatne zaplete, saj se moramo odločiti glede postopka diskretizacije, ki lahko pomembno vpliva na izračun samih informacijskih mer. Druga možnost je, da uporabimo diferencialne informacijsko-teoretične mere, pri katerih je potrebno oceniti gostoto verjetnosti in izvesti numerično integracijo. Obstajajo tudi drugi pristopi za ocenjevanje informacijskih mer, ki ne uporabljajo ocenjevanja verjetnostnih porazdelitev, vendar so vezani na specifično informacijsko mero [20].

Erdogmus in sod. so se v svojem delu [25] osredotočili na kvadratno Renyijevo entropijo in izpeljali postopek za njen izračun oziroma oceno neposredno iz podatkov, brez vmesnega ocenjevanja gostote verjetnosti. Idejo so razširili še na koncept medsebojne informacije in na podlagi Cauchy-Schwarzeve neenakosti predlagali kvadratno medsebojno informacijo [7]. Dokazali so, da je kvadratna medsebojna informacija enaka nič, ko sta naključni spremenljivki neodvisni, in pozitivna v primeru, da med njima obstaja neka odvisnost. Prav tako so empirično pokazali, da minimuma in maksimuma Shannonove in kvadratne medsebojne informacije sovpadata, vendar sta različna v magnitudi. Koncept kvadratne medsebojne informacije so razširili tudi na več naključnih spremenljivk.

B.3 Izbira značilk

Značilka je izmerjena lastnost opazovanega sistema. Opisujejo jo lahko surovi podatki, pridobljeni ob opazovanju sistema, ali pa so ti pripravljeni tako, da je nadaljnja analiza čim enostavnejša. Nad podatki lahko potem uporabimo metode strojnega učenja, ki omogočajo gradnjo napovednih modelov in interpretacijo podatkov ter s tem boljše razumevanje delovanja opazovanega sistema. Podatki pogosto vsebujejo zelo veliko značilk, ki so nepomembne pri postavitvi modela sistema. Gradnja modela na podlagi velikega števila značilk lahko vodi k povečanemu času procesiranja podatkov in

neustreznemu posploševanju znanja. Da se temu izognemo, je priporočljivo zmanjšati količino značilk. Tu sta možna dva pristopa: gradnja značilk transformira osnovni nabor značilk v manjšega, pri tem pa poskuša ohraniti čim več informacij uporabnih pri gradnji modela; izbira značilk pa po drugi strani poskuša v osnovnem naboru značilk poiskati tiste, ki nosijo največ informacij relevantnih za gradnjo modela. V principu je možna tudi kombinacija obeh pristopov.

Značilke v splošnem ločimo v tri kategorije: pomembne, pogojno-pomembne in nepomembne. V splošnem si želimo v končne naboru značilk imeti le pomembne značilke, ki vsebujejo vso informacijo, potrebno za gradnjo napovednega modela. Pogojno-pomembne značilke so lahko sicer koristne, vendar je informacija, ki jo nosijo, že vsebovana v pomembnih značilkah.

Proces izbire značilk lahko kategoriziramo glede na štiri vidike [35]:

- Smer iskanja, ki definira, na kak način se značilke dodajajo ali odvijajo iz nabora izbranih značilk.
- Strategija iskanja, ki definira, na kak način bomo preiskali prostor naborov značilk. Izčrpno preiskovanje celotnega prostora običajno ni sprejemljivo, zato se tu pojavljajo razne požrešne strategije preiskovanja.
- Ocenjevalni kriterij, ki podaja oceno pomembnosti posameznega nabora značilk.
- Ustavitveni kriterij, ki pove, kdaj se proces izbire značilk zaključuje.

Metode za izbiro značilk lahko razdelimo v tri velike skupine [35]: metode z ovojnico, vgrajene metode, metode z uporabo sita. Metode z ovojnico uporabljajo uspešnost zgrajenega napovednega modela za oceno pomembnosti nabora izbranih značilk. Te metode so običajno računsko zelo zahtevne, vendar lahko značilke izbrane na ta način znatno izboljšajo uspešnost napovednega modela. Vgrajene metode so hitrejša, vendar so vezane na specifični učni postopek, ki se uporablja za gradnjo modela, kar omejuje njihovo uporabnost. Zadnja skupina metod za izbiro značilk so metode z uporabo sita. Te so ločene od učnega postopka, ki gradi napovedni model in se uporabljajo za predprocesiranje vhodne podatkovne zbirke. Kot kriterij za izbiro značilk uporabljajo različne mere kot so korelacija [47] in medsebojna informacija [48].

Ravno informacijske mere so zaradi svojih lastnosti med bolj uporabljanimi kriteriji za izbiro značilk v metodah z uporabo sita [52]. Tekom razvoja področja izbire značilk je bilo predlaganih veliko metod, na primer informacijski prispevek [29], ki razvrsti značilke glede na informacijo, ki si jo delijo z želenim izhodom napovednega modela. Potem so tu metode prvega reda, na primer MIFS in MRMR [3], ki poskušajo pri izbiri značilk upoštevati odvisnosti med značilkami ter metode drugega reda, na primer JMI [57], ki omogoča zaznavanje interakcij med izhodom in pari značilk. Obstajajo seveda tudi metode višjih redov, ki ocenjujejo pomembnost celotnih naborov značilk; pri teh hitro nastane težava ocenjevanja verjetnostnih porazdelitev v visokih dimenzijah.

B.4 Predlagana metoda

Predlagana metoda QMIFS sloni na kvadratni medsebojni informaciji kot kriteriju za izbiro pomembnih značilk. Motivacija za novo metodo je bila možnost izračuna kvadratne medsebojne informacije neposredno iz podatkov, tako za diskretne kot tudi za zvezne značilke. To odstrani potrebo po dodatnem predprocesiranju podatkov. Prav tako se metoda izogne uporabi parametrov, ki bi otežili njeno uporabo. QMIFS zmere med značilkami zaznati interakcije drugega reda, kar je primerljivo z metodo JMI. Pri preiskovanju prostora značilk uporabljamo iterativno požrešno iskalno strategijo, ki v nabor izbranih značilk dodaja nove značilke eno za drugo, dokler ne dosežemo kriterija za ustavitev postopka.

Ta je v našem primeru kar vnaprej določeno število značilk. Na vsakem koraku algoritma kot kriterij za izbiro nove značilke uporabimo heuristiko, ki upošteva kvadratno medsebojno informacijo med vsemi možnimi trojicami (že izbrana značilka, značilka kandidatka, izhod) ter kvadratno informacijo med vsemi pari značilk kandidat in že izbranih značilk, ki ocenjuje podobnost/odvisnost značilk med sabo. V končnem naboru značilk želimo namreč imeti čim manj med sabo močno podobnih značilk. Z vsako značilko kandidatko izračunamo vsoto razlik prej omenjenih količin preko vseh že izbranih značilk. Značilka kandidatka z najvišjo vrednostjo heuristike se nato doda v nabor že izbranih značilk.

Delovanje metode QMIFS smo preverili na umetnih in realnih podatkovnih zbirkah. Najprej smo metodo preizkusili na treh umetnih podatkovnih zbirkah, v katerih so pomembne značilke poznane že vnaprej. Izkaže se, da se metoda obnaša po pričakovanjih. V podatkih uspe odkriti odvisnosti drugega reda. Pri tem je omejena s

požrešno iskalno strategijo, ki prostor značilk preiskuje iterativno. To lahko povzroči, da metoda izbere pogojno-pomembne značilke ali pa spregleda interakcija drugega reda v primeru, ko nobena od značilk ne izkazuje vsaj delne povezanosti z izhodom, če jo opazujemo samostojno.

Preverjanje na realnih podatkovnih zbirkah je bilo opravljeno posredno preko gradnje napovednega modela. Izbrali smo 16 podatkovnih zbirk različnih obsegov in problemskih domen. Preko uspešnosti napovednega modela, zgrajenega na podlagi izbranih značilk s pomočjo klasifikacijskega drevesa ali regresijskega drevesa (odvisno od problemske domene), smo metodo QMIFS primerjali z ostalimi primerljivimi informacijsko-teoretičnimi metodami. V primerjavo so bile vključene metode MIFS, MRMR, JMI in ReliefF, ki edina od navedenih ne sloni na informacijskih merah. Uspešnost zgrajenega napovednega modela smo v primeru uvrščanja merili s pomočjo klasifikacijske točnosti, površine pod krivuljo in Youdenovega indeksa. V primeru regresije pa smo uspešnost merili s korenem povprečne kvadratne napake.

Metoda je na problemih uvrščanja primerljiva z ostalimi in se glede na rangiranje na podlagi uspešnosti uvršča nekje v sredino. Je boljša od MIFS, MRMR in ReliefF, vendar zaostaja za metodo JMI. Pri regresiji se, vsaj na uporabljenih podatkovnih zbirkah, izkaže kot v splošnem boljša od preostalih metod. Kljub temu test ni pokazal statistično signifikantnih razlik med metodami. Za sklepanje na statistično signifikantnost rezultatov bi potrebovali veliko večje število podatkovnih zbirk.

B.5 Izboljšave QMIFS

Računska zahtevnost metod za izbiro značilk lahko postane problematična, kadar imamo opravka z zelo velikimi podatkovnimi zbirkami, kar privede do zelo dolgih izvajalnih časov. Te lahko zmanjšamo na dva načina. Poskušamo lahko zmanjšati računsko zahtevnost algoritma s pomočjo približnih metod ali spretne uporabe podatkovnih struktur. Druga možnost je, da uporabimo masovno-paralelne arhitekture kot so grafične procesne enote ali računski koprocesorji. Tu predstavljamo obe možnosti v okviru izboljšav algoritma QMIFS.

V [84] so avtorji predstavili način, kako s pomočjo nepopolne dekompozicije Choleskega zmanjšamo računsko zahtevnost osnovnega algoritma za izračun kvadratne medsebojne informacije. Njihov pristop k računanju kvadratne medsebojne informacije smo prilagodili za izračun kriterijske funkcije metode QMIFS. Izpeljali smo razširitev osnovne metode, ki deluje le za dve spremenljivki, na tri spremenljivke ter dosegli

zmanjšanje računske zahtevnosti. Izkaže se, da je čas izvajanja pri tej metodi močno odvisen od narave podatkov. V teoriji se lahko zgodi, da metoda ni nič hitrejša od naivnega računanja ali je celo počasnejša, vendar se v praksi to običajno ne dogaja.

Pohitritve metode QMIFS smo se lotili tudi z boljšim izkoriščanjem strojne opreme, ki je na voljo v sodobnih računalnikih. Grafične procesne enote nudijo veliko računске zmogljivosti in niso več omejene le na procesiranje grafike. Proizvajalci grafičnih procesnih enot nudijo razvojna okolja, kot je CUDA [9], ki omogoča relativno enostaven pristop k programiranju grafičnih procesnih enot za potrebe splošno-namenskega računanja. Poleg grafičnih procesnih enot so se na trgu pojavili specializirani računski koprocesorji, kot je na primer Intelov Xeon Phi osnovan na arhitekturi MIC (ang. many integrated core architecture) [108]. Obe rešitvi spadata med tako imenovane masovno-paralelne arhitekture. To pomeni, da morajo biti problemi, ki jih hočemo z njimi reševati, takšni, da je mogoče delo zelo na drobno porazdeliti. V nasprotnem primeru ne dosežemo zelenih rezultatov.

Na primeru metode QMIFS smo algoritem prilagodili tako za grafične procesne enote kot tudi za koprocesorje Xeon Phi. Pri tem smo bili pazljivi, da smo upoštevali specifične posamezne arhitekture in čim bolj zmanjšali režijo. Raziskali smo tudi možnost uporabe obeh arhitektur in več-jedrne centralne procesne enote hkrati in s tem poskušali še dodatno zmanjšati čas procesiranja.

Izmerili smo čase procesiranja za različne implementacije metode QMIFS (naivna, uporaba nepopolne dekompozicije Choleskega, GPU, MIC) na vseh 16 podatkovnih zbirkah in jih primerjali z ostalimi metodami (MIFS, MRMR, JMI in Relief). Rezultati kažejo, da vse tri izboljšave močno pripomorejo k zmanjšanju časa izvajanja v primerjavi z naivno metodo in omogočijo uporabo QMIFS tudi na večjih podatkovnih zbirkah. V primerjavi z ostalimi informacijsko-teoretičnimi metodami se izkaže, da je pri problemih uvrščanja QMIFS večinoma hitrejši. Razlog za to je v dodatnem predprocesiranju (diskretizaciji), ki je potrebna pri ostalih metodah. Tu je bila uporabljena napredna metoda MDL [79]. Na regresijskih problemih je bila po drugi strani uporabljena preprostejša oblika diskretizacije – razvrščanje na osnovi enake frekvence. Ta je računsko precej manj zahtevna, zato so v tem primeru implementacije QMIFS nekoliko počasnejše od preostalih informacijsko-teoretičnih metod. V obeh primerih pa je bila najhitrejša metoda Relief, ki edina ne temelji na informacijski teoriji in ima zelo nizko računsko zahtevnost.

Nadalje smo različne implementacije QMIFS preizkusili na umetnih podatkih, da

bi ugotovili, kako se obnašajo pri spreminjanju števila značilk in primerov v podatkih. Grafična procesna enota se obnese najbolje, saj za skoraj stokrat zmanjša čas procesiranja v primerjavi z naivno implementacijo. Na koprocesorju je pohitritev okoli šestdesetkratna, medtem ko je pohitritev, ki jo prinese nepopolna dekompozicija Choleskega, okoli dvajsetkratna. Eksperiment z uporabo vseh procesnih enot računalnika pokaže, da je ta pristop obetaven, saj na primeru, ko uporabimo 12 jeder centralne procesne enote, grafično procesno enoto ter koprocesor Xeon Phi, znatno zmanjšamo čas izvajanja metode QMIFS.

B.6 Zaključek

V disertaciji smo se osredotočili na informacijsko-teoretični pristop k izbiri značilk. Povzeli smo temelje teorije informacij in si ogledali njeno posplošitev. Prav tako smo raziskali področje izbiranja pomembnih značilk in njegovo povezavo s teorijo informacij. Na podlagi pregleda vseh teh področij smo predlagali novo metodo za izbiro značilk, QMIFS, ki sloni na kvadratni medsebojni informaciji. Naš cilj je bil razvoj metode, ki bo pri izbiranju značilk upoštevala interakcije med njimi in bo sposobna analize tako diskretnih kot tudi zveznih značilk brez predprocesiranja/diskretizacije podatkov. Prav tako smo se izognili uporabi parametrov, ki bi laikom lahko otežili uporabo metode.

Delovanje metode smo analizirali na umetnih in realnih podatkih ter jo primerjali z drugimi metodami za izbiro značilk. Na realnih podatkovnih zbirkah smo metode med sabo primerjali posredno, preko uspešnosti napovednega modela zgrajenega s pomočjo klasifikacijskega oziroma regresijskega drevesa. Rezultati kažejo, da je naša metoda primerljiva z ostalimi pri iskanju značilk na problemski domeni uvrščanja, pri regresiji pa se v splošnem izkaže za boljšo.

Prav tako smo izboljšali računsko zahtevnost metode s pomočjo nepopolne dekompozicije Choleskega in dosegli precejšnje prihranke pri izvajalnem času. Nadalje smo raziskali možnost implementacije metode QMIFS na masovno-paralelnih arhitekturah, kot so grafične procesne enote in računski koprocesorji. Implementacija na obeh arhitekturah se je izkazala za uspešno, saj močno zmanjša izvajalni in naredi metodo QMIFS uporabno tudi na večjih podatkovnih zbirkah.

Naša metoda se izkaže kot univerzalna, primerna tako za izbiro značilk pri problemih uvrščanja in regresije. Ne potrebuje predprocesiranja in se izogne uporabi parametrov. Skozi dodatne izboljšave, kot so uporaba nepopolne dekompozicije Choleskega

ali masovno-paralelnih arhitektur, pa postane uporabna tudi za procesiranje velikih podatkovnih zbirk.

Za nadaljnje delo ostaja odprtih več front. Ena od teh je optimizacija širine jedrne funkcije, ki se uporablja pri računanju kvadratne medsebojne informacije, saj pomembno vpliva na končno vrednost mere. Druga možnost je prilagoditev našega pristopa na kako drugo informacijsko-teoretično mero, ki bi lahko privedla do še bolj-
ših rezultatov. Za nadaljnje delo se ponuja tudi možnost implementacije nepopolne dekompozicije Choleskega na masovno-paralelnih arhitekturah, ki zaradi težavne optimizacije in prilagoditve algoritma ostaja odprt problem.



BIBLIOGRAPHY

- [1] Michel Verleysen, Fabrice Rossi, and Damien François. Advances in feature selection with mutual information. In *Similarity-Based Clustering*, pages 52–69. Springer, 2009.
- [2] Manoranjan Dash and Huan Liu. Feature selection for classification. *Intelligent data analysis*, 1(1-4): 131–156, 1997.
- [3] Nojun Kwak and Chong-Ho Choi. Input feature selection for classification problems. *IEEE Transactions on Neural Networks*, 13(1):143–159, 2002.
- [4] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 2012. ISBN 9781118585771.
- [5] Kenneth E Hild, Deniz Erdogmus, Kari Torkkola, and Jose C Principe. Feature extraction using information-theoretic learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9): 1385–1392, 2006.
- [6] Shigeru Furuichi. Information theoretical properties of tsallis entropies. *Journal of Mathematical Physics*, 47(2):023302, 2006.
- [7] Jose C Principe. *Information theoretic learning: Renyi's entropy and kernel perspectives*. Springer Science & Business Media, New York, USA, 2010.
- [8] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5): 879–899, 2008.
- [9] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008.
- [10] Yvan Saeys, Inaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517, 2007.
- [11] J.C.A. van der Lubbe. *Information Theory*. Cambridge University Press, 1997. ISBN 9780521467605.
- [12] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.
- [13] A. Dobnikar. *Teorija informacij in sistemov*. Založba FE in FRI, 2009. ISBN 9789616209717.
- [14] Aleks Jakulin. *Machine learning based on attribute interactions*. PhD thesis, Univerza v Ljubljani, 2005.
- [15] Satoshi Watanabe. Information theoretical analysis of multivariate correlation. *IBM Journal of research and development*, 4(1):66–82, 1960.
- [16] Christian Beck. Generalised information and entropy measures in physics. *Contemporary Physics*, 50(4): 495–510, 2009.
- [17] Alfred Renyi. On measures of entropy and information. In *Fourth Berkeley symposium on mathematical statistics and probability*, pages 547–561, 1961.
- [18] Salvador Garcia, Julian Luengo, José Antonio Sáez, Victoria Lopez, and Francisco Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750, 2013.
- [19] Keki B Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1029, Chambéry, France, 1993. Morgan-Kaufmann.
- [20] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.
- [21] Janett Walters-Williams and Yan Li. Estimation of mutual information: A survey. In *International Conference on Rough Sets and Knowledge Technology*, pages 389–396. Springer, 2009.

- [22] Masashi Sugiyama. Machine learning with squared-loss mutual information. *Entropy*, 15(1):80–112, 2012.
- [23] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.
- [24] Vladimir Katkovnik and Ilya Shmulevich. Kernel density estimation with adaptive varying window size. *Pattern recognition letters*, 23(14):1641–1648, 2002.
- [25] Deniz Erdogmus and Jose C Principe. Generalized information potential criterion for adaptive system training. *IEEE Transactions on Neural Networks*, 13(5):1035–1044, 2002.
- [26] W. Rudin. *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill, 1976. ISBN 9780070856134.
- [27] S Rao. *Unsupervised Learning: An Information Theoretic Learning Approach*. PhD thesis, University of Florida, Gainesville, 2008.
- [28] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.
- [29] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.
- [30] Jorge R Vergara and Pablo A Estévez. A review of feature selection methods based on mutual information. *Neural Computing and Applications*, 24(1):175–186, 2014.
- [31] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.
- [32] Kari Torkkola. Feature extraction by non-parametric mutual information maximization. *Journal of machine learning research*, 3(Mar):1415–1438, 2003.
- [33] Steve De Backer, Antoine Naud, and Paul Scheunders. Non-linear dimensionality reduction techniques for unsupervised feature extraction. *Pattern Recognition Letters*, 19(8):711–720, 1998.
- [34] Sangita Sharma, Dan Ellis, Sachin Kajarekar, Pratibha Jain, and Hynek Hermansky. Feature extraction using non-linear transformation for robust speech recognition on the Aurora database. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 2, pages II1117–II1120. IEEE, 2000.
- [35] Avrim L Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271, 1997.
- [36] George H John, Ron Kohavi, Karl Pfleger, et al. Irrelevant features and the subset selection problem. In *Machine learning: proceedings of the eleventh international conference*, pages 121–129, 1994.
- [37] Luis Carlos Molina, Luis Belanche, and Àngela Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 306–313. IEEE, 2002.
- [38] Hervé Stoppiglia, Gérard Dreyfus, Rémi Dubois, and Yacine Oussar. Ranking a random feature for variable and feature selection. *Journal of machine learning research*, 3(Mar):1399–1414, 2003.
- [39] Amir Navot, Ran Gilad-Bachrach, Yiftah Navot, and Naftali Tishby. Is feature selection still necessary? In *Subspace, latent structure and feature selection*, pages 127–138. Springer, 2006.
- [40] Ron Kohavi and George H John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1):273–324, 1997.
- [41] David W Aha and Richard L Bankert. A comparative evaluation of sequential feature selection algorithms. In *Learning from Data*, pages 199–206. Springer, 1996.
- [42] Pavel Pudil, Jana Novovičová, and Josef Kittler. Floating search methods in feature selection. *Pattern recognition letters*, 15(11):1119–1125, 1994.
- [43] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Artificial Intelligence. Addison-Wesley Publishing Company, 1989. ISBN 9780201157673.
- [44] James Kennedy. Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer, 2011.
- [45] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
- [46] J Ross Quinlan. *C4.5: programs for machine learning*. Elsevier, 2014.
- [47] Mark A Hall. Correlation-based feature selection of discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 359–366, 2000.
- [48] François Fleuret. Fast binary feature selection with conditional mutual information. *Journal of Machine Learning Research*, 5(Nov):1531–1555, 2004.
- [49] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

- [50] Kenji Kira and Larry A Rendell. A practical approach to feature selection. In *Proceedings of the ninth international workshop on Machine learning*, pages 249–256, 1992.
- [51] Hussein Almuallim and Thomas G Dietterich. Learning with many irrelevant features. In *AAAI*, volume 91, pages 547–552, 1991.
- [52] Patrick Emmanuel Meyer, Colas Schretter, and Gianluca Bontempi. Information-theoretic feature selection in microarray data using variable complementarity. *IEEE Journal of Selected Topics in Signal Processing*, 2(3):261–274, 2008.
- [53] Roberto Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, 5(4):537–550, 1994.
- [54] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.
- [55] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. *Journal of Machine Learning Research*, 13 (Jan):27–66, 2012.
- [56] Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2):189–201, 2009.
- [57] H Yang and John Moody. Feature selection based on joint mutual information. In *Proceedings of international ICSC symposium on advances in intelligent data analysis*, pages 22–25, 1999.
- [58] Leonardo Barroso Gonçalves and José Leonardo Ribeiro Macrini. Rényi entropy and cauchy-schwarz mutual information applied to MIFS-U variable selection algorithm: a comparative study. *Pesquisa Operacional*, 31(3):499–519, 2011.
- [59] Can-Tao Liu and Bao-Gang Hu. Mutual information based on Renyi's entropy feature selection. In *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, volume 1, pages 816–820. IEEE, 2009.
- [60] Davor Sluga and Uroš Lotrič. Generalized information-theoretic measures for feature selection. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 189–197. Springer, 2013.
- [61] Tommy WS Chow and D Huang. Estimating optimal feature subsets using efficient estimation of high-dimensional mutual information. *IEEE Transactions on Neural Networks*, 16(1):213–224, 2005.
- [62] Kanaka Rajan and William Bialek. Maximally informative “stimulus energies” in the analysis of neural responses to natural signals. *PLOS one*, 8(11):e71959, 2013.
- [63] Jeffrey D Fitzgerald, Ryan J Rowekamp, Lawrence C Sincich, and Tatyana O Sharpee. Second order dimensionality reduction using minimum and maximum mutual information models. *PLoS Comput Biol*, 7(10):e1002249, 2011.
- [64] Ryan J Rowekamp and Tatyana O Sharpee. Analyzing multicomponent receptive fields from neural responses to natural stimuli. *Network: Computation in Neural Systems*, 22(1-4):45–73, 2011.
- [65] Noelia Sánchez-Marofío, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter methods for feature selection—a comparative study. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 178–187. Springer, 2007.
- [66] Jaesung Lee and Dae-Won Kim. Fast multi-label feature selection based on information-theoretic feature ranking. *Pattern Recognition*, 48(9):2761–2771, 2015.
- [67] Benoît Frénay, Gauthier Doquire, and Michel Verleysen. Is mutual information adequate for feature selection in regression? *Neural Networks*, 48:1–7, 2013.
- [68] Davor Sluga and Uroš Lotrič. Quadratic mutual information feature selection. *Entropy*, 19(4):157, 2017.
- [69] Juha Reunanen. Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3(Mar):1371–1382, 2003.
- [70] M. Lichman. UCI machine learning repository. <http://archive.ics.uci.edu/ml>, 2013.
- [71] Daniela Witten, Robert Tibshirani, Sam Guoping Gu, Andrew Fire, and Weng-Onn Lui. Ultra-high throughput sequencing-based small rna discovery and discrete statistical biomarker analysis in a collection of cervical tumours and matched controls. *BMC Biology*, 8(1):58, May 2010. doi: [10.1186/1741-7007-8-58](https://doi.org/10.1186/1741-7007-8-58).
- [72] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999. doi: [10.1126/science.286.5439.531](https://doi.org/10.1126/science.286.5439.531).

- [73] Kamaljit Singh, Ranjeet Kaur, and Dinesh Kumar. Comment volume prediction using neural networks and decision trees. In *Proceedings of the 2015 17th UKSIM-AMSS International Conference on Modelling and Simulation*, UKSIM '15, pages 15–20, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4799-8713-9. doi: [10.1109/UKSim.2015.20](https://doi.org/10.1109/UKSim.2015.20).
- [74] Krisztian Buza. Feedback prediction for blogs. In *Data analysis, machine learning and knowledge discovery*, pages 145–152. Springer, 2014.
- [75] C. Rasmussen. Data for evaluating learning in valid experiments. <http://www.cs.toronto.edu/~dave/>, 1996.
- [76] Gavin Brown. A new perspective for information theoretic feature selection. In *AISTATS*, pages 49–56, 2009.
- [77] Marko Robnik-Šikonja and Igor Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine Learning*, 53(1):23–69, Oct 2003. ISSN 1573-0565. doi: [10.1023/A:1025667309714](https://doi.org/10.1023/A:1025667309714).
- [78] Yi Zhang, Chris Ding, and Tao Li. Gene selection algorithm by combining relief and mrmr. *BMC genomics*, 9(2):S27, 2008.
- [79] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
- [80] Ahmad Ashari, Iman Paryudi, and A Min Tjoa. Performance comparison between naïve bayes, decision tree and k-nearest neighbor in searching alternative design in an energy simulation tool. *Int. J. Adv. Comput. Sci. Appl.*, 4(11):33–39, 2013.
- [81] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [82] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.
- [83] Aydın Ulaş, Olcay Taner Yildiz, and Ethem Alpaydın. Cost-conscious comparison of supervised learning algorithms over multiple data sets. *Pattern Recognition*, 45(4):1772–1781, 2012.
- [84] Sohan Seth and José C Príncipe. On speeding up computation in information theoretic learning. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 2883–2887. IEEE, 2009.
- [85] Shai Fine and Katya Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2(Dec):243–264, 2001.
- [86] Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig. Accelerating molecular dynamics simulations using graphics processing units with CUDA. *Computer Physics Communications*, 179(9):634–641, 2008.
- [87] Lorenzo Dematté and Davide Prandi. GPU computing for systems biology. *Briefings in bioinformatics*, 11(3):323–333, 2010.
- [88] Anders Eklund, Paul Dufort, Daniel Forsberg, and Stephen M LaConte. Medical image processing on the GPU—past, present and future. *Medical image analysis*, 17(8):1073–1094, 2013.
- [89] Liheng Jian, Cheng Wang, Ying Liu, Shenshen Liang, Weidong Yi, and Yong Shi. Parallel data mining techniques on graphics processing unit with compute unified device architecture (CUDA). *The Journal of Supercomputing*, 64(3):942–967, 2013.
- [90] André R Brodtkorb, Trond R Hagen, and Martin L Setra. Graphics processing unit (GPU) programming strategies and trends in GPU computing. *Journal of Parallel and Distributed Computing*, 73(1):4–13, 2013.
- [91] John Von Neumann. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.
- [92] David Luebke and Greg Humphreys. How GPUs work. *Computer*, 40(2), 2007.
- [93] Nicholas Wilt. *The CUDA handbook: A comprehensive guide to GPU programming*. Pearson Education, 2013.
- [94] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, 28(2), 2008.
- [95] Jonathan Cohen and Michael Garland. Solving computational problems with GPU computing. *Computing in Science & Engineering*, 11(5):58–63, 2009.
- [96] John E Stone, David Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66–73, 2010.
- [97] Felix Weninger, Johannes Bergmann, and Björn W Schuller. Introducing current: the munich open-source CUDA recurrent neural network toolkit. *Journal of Machine Learning Research*, 16(3):547–551, 2015.
- [98] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on GPUs with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, volume 3. Citeseer, 2011.

- [99] Noel Lopes and Bernardete Ribeiro. GPUMLib: An efficient open-source GPU machine learning library. *International Journal of Computer Information Systems and Industrial Management Applications*, 3:355–362, 2011.
- [100] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU math compiler in python. In *Proc. 9th Python in Science Conf.*, pages 1–7, 2010.
- [101] Ling Sing Yung, Can Yang, Xiang Wan, and Weichuan Yu. GBOOST: a GPU-based tool for detecting gene–gene interactions in genome-wide case control studies. *Bioinformatics*, 27(9):1309–1310, 2011.
- [102] Ivo D Shterev, Sin-Ho Jung, Stephen L George, and Kourous Owzar. permGPU: Using graphics processing units in RNA microarray association studies. *BMC bioinformatics*, 11(1):329, 2010.
- [103] Yong-Yeon Jo, Sang-Wook Kim, and Duck-Ho Bae. Efficient sparse matrix multiplication on GPU for large social network analysis. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1261–1270. ACM, 2015.
- [104] Fatemeh Azmandian, Ayse Yilmazer, Jennifer G Dy, Javed A Aslam, and David R Kaeli. GPU-accelerated feature selection for outlier detection using the local kernel density ratio. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 51–60. IEEE, 2012.
- [105] Alberto Guillén, M Isabel García Arenas, Mark van Heeswijk, Dusan Sovilj, Amaury Lendasse, Luis Javier Herrera, Héctor Pomares, and Ignacio Rojas. Fast feature selection in a GPU cluster using the delta test. *Entropy*, 16(2):854–869, 2014.
- [106] Davor Sluga, Tomaž Curk, Blaž Zupan, and Uroš Lotrič. Acceleration of information-theoretic data analysis with graphics processing units. *Przegląd Elektrotechniczny*, 88(2):136–139, 2012.
- [107] Davor Sluga, Tomaž Curk, Blaž Zupan, and Uroš Lotrič. Heterogeneous computing architecture for fast detection of SNP-SNP interactions. *BMC bioinformatics*, 15(1):216, 2014.
- [108] Erik Saule, Kamer Kaya, and Ümit V Çatalyürek. Performance evaluation of sparse matrix multiplication kernels on intel Xeon Phi. In *International Conference on Parallel Processing and Applied Mathematics*, pages 559–570. Springer, 2013.
- [109] Alexander Heinecke, Michael Klemm, and Hans-Joachim Bungartz. From gpgpu to many-core: Nvidia Fermi and intel many integrated core architecture. *Computing in Science & Engineering*, 14(2):78–83, 2012.
- [110] TOP500 supercomputer site, 2017. URL <http://www.top500.org>.
- [111] Xing Liu, Mikhail Smelyanskiy, Edmond Chow, and Pradeep Dubey. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 273–282. ACM, 2013.
- [112] Tao Gao, Yutong Lu, Baida Zhang, and Guang Suo. Using the Intel many integrated core to accelerate graph traversal. *The International Journal of High Performance Computing Applications*, 28(3):255–266, 2014.
- [113] Tim Cramer, Dirk Schmidl, Michael Klemm, and Dieter an Mey. OpenMP programming on Intel Xeon Phi coprocessors: An early performance comparison. In *Proc. Many Core Appl. Res. Community (MARC) Symp.*, pages 38–44, 2012.
- [114] Lei Jin, Zhaokang Wang, Rong Gu, Chunfeng Yuan, and Yihua Huang. Training large scale deep neural networks on the Intel Xeon Phi many-core coprocessor. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 1622–1630. IEEE, 2014.
- [115] André Viebke and Sabri Pllana. The potential of the Intel Xeon Phi for supervised deep learning. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferece on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, pages 758–765. IEEE, 2015.